

# **A User Interface for *Anusaaraka***

*A Thesis Submitted*  
*in Partial Fulfillment of the Requirements*  
*for the Degree of*  
**MASTER OF TECHNOLOGY**

by

**Sanjil Saxena**



**Department of Computer Science and Engineering,**  
**Indian Institute of Technology Kanpur**

*February, 1998*

1 MAY 1998

CENTRAL LIBRARY  
117  
No. 125386

GSE-1998-M-SAX-USE

Entered in system

Ans  
4-5-98



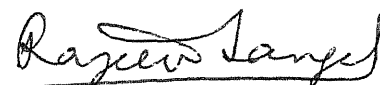
A125386

# CERTIFICATE

This is to certify that the thesis work entitled "**A User Interface for Anusaaraka**" by **Sanjil Saxena** has been carried out under my supervision and has not been submitted elsewhere for a degree.



(Prof. T. V. Prabhaker)  
Dept. of Comp. Sc. & Engg.  
I. I. Kanpur  
India



(Prof. Rajeev Sangal)  
Dept. of Comp. Sc & Engg.  
I. I. T. Kanpur  
India

Feb. 1998

# ACKNOWLEDGMENTS

With great pleasure and deep sense of gratitude, I acknowledge the invaluable guidance of Prof. Rajeev Sangal, my thesis supervisor, In spite of his extremely busy schedule, he could find time to provide precious guidance and that extra bit of support, required for the completion of this thesis. His never ending patience and ever smiling face was the source of inspiration for me to complete this work.

I express sincere thanks to Prof. T. V. Prabhakar, who took pains and managed all the official things needed for submitting the thesis and defense.

I express my deep sense of indebtedness to Dr. Vineet Chaitanya for the continued support he has provided throughout this work. He was always there to help me and put me on the right track which led to smooth finish of my thesis. His constructive criticism have had a great bearing on the form and content of this thesis.

I express my sincere gratitude to Prof. G U Rao for his keen interest in my work and constant support given during my stay at Center University Hyderabad.

Many people have helped me in carrying out my work smoothly. I express my thanks to Ms Amba Kulkarni for helping me many times patiently, in spite of having heavy load of work on her. I thank Dr. Pushpak Bhattacharya (I. I. T. Bombay) for helping me in the implementation of some heuristics for improving the readability of the output. I thank Mr. Mahash Ghankot for providing valuable suggestions for the improvement of the post-editing interface, and for helping me in preparing examples to be quoted in the thesis report.

I express my deep regards to Nisha didi, and love dear Sapna and Gonu, who never let me feel that I was away from my home.

Last, but not the least, I express my heart felt regards to Ms. Ruchi Goel who provided me support and inspiration through her mails throughout my stay at Hyderabad.

# Abstract

This thesis designs and develops an interface for *Amusaaraka*. The *Amusaaraka* is an attempt at overcoming the language barrier within Indian Languages with the help of machine. In the *amusaaraka* model, the text in source language is first pre-edited, then it is given to *amusaaraka* as input. *Amusaaraka* produces output, which is in a dialect of Hindi call it say, *amumarit* Hindi. This output can be read directly by a trained user, or it can be post-edited by a human post-editor and made available to a normal reader. The user can interact with the *amusaaraka* system at three different points, consequently the interface for *amusaaraka* has three subparts: pre-editing interface, reading interface and post-editing interface.

In pre-editing, the source language text is checked and corrected against typos, undesirable *sandhis*, and dialectal spellings. A human pre-editor corrects these. The pre-editing interface helps the pre-editor in identifying unrecognized words, and in performing and splitting *sandhis*.

The reading interface helps the user in reading the *amusaaraka* output. It presents the output in a suitable form, hiding certain details, however these details could be made available to the user on the click of a button. The Reading interface also provides the user with online context-sensitive help. It provides help regarding the notations used in the output, and brief help as well as detailed tutorials on the artificial words introduced in the output. In addition it also provides the dictionary help i.e. the user can look into the dictionary corresponding to the target language word. The Reading interface also provides help on the words which are not recognized by the machine. This help includes telling the *vibhakti* of the unrecognized word, and dictionary look-up for words which have approximately same spelling.

The post-editing interface facilitates the post-editing of the *amusaaraka* output by the human post-editor. It provides tools which perform commonly needed transformations for converting the output text into the grammatically correct target language text, under the instruction of the user. The post-editing commands reduces the task of post-editing from typing and re-structuring the whole sentence to typing a few commands. The post-editing interface can also have a target language thesaurus integrated with it. The availability of the online thesaurus helps the user in selecting the appropriate synonym for the target language word. If appropriate synonym is not available, the user, through the interface can enter it into the thesaurus so that it will be there for use in future.

# Contents

<b>1. Introduction</b>	<b>1</b>
1.1 Anusaaraka Machine Translation Model	1
1.2 Pre-editing	2
1.3 Reading the output	3
1.4 Post-editing	6
1.5 Objectives of the system	7
1.6 Issues in a good interface design	8
1.7 Outline of the thesis	9
<b>2. The Anusaaraka</b>	<b>11</b>
2.1 Introduction	11
2.2 The Anusaaraka Solution	11
2.2.1 Sharing the load	
2.2.2 Faithfulness vs. naturalness	
2.3 Anusaaraka Output Language	12
2.3.1 Artificial Words	
2.3.2 Language bridges	
2.3.2.1 "ki" Construction	
2.3.2.2 "jO" construction	
2.3.2.3 "nE" construction	
2.3.3 Agreement	
2.3.4 Multiple Options	
2.3.5 Extra Information	
2.4 Anusaaraka Claim	20
<b>3. Pre-editing Interface</b>	<b>21</b>
3.1 Introduction	21

3.2 Why Standardization in a language? .....	21
3.3 How Standardization helps anusaaraka .....	22
3.4 Using the Pre-Editing Interface .....	24
<b>4. General Reading Interface .....</b>	<b>26</b>
4.1 Controlling amount of information .....	26
4.2 On Line Help .....	28
4.2.1 Help on Notation	
4.2.2 Help on Words with 'Back-quote'	
4.2.3 Tutorial on Words	
4.2.3 Thesaurus Help	
4.3 Unrecognized Words .....	31
4.3.1 Foreign Words	
4.3.2 Unrecognized Words due to incomplete Morph	
<b>5. Post-Editing Interface .....</b>	<b>33</b>
5.1 Introduction .....	33
5.2 Gender Number Person Agreement .....	34
5.3 Noun Phrase Agreement .....	35
5.3.1 Adding Vibhakti	
5.3.2 Giving Gender	
5.4 Selecting from Multiple options .....	37
5.5 Handling jO_* Construction .....	38
5.6 Miscellaneous Commands .....	39
5.6.1 Heuristic for plurals	
5.6.2 Filling words by the user	
5.6.3 Giving Gender Number Information by the user	
5.6.4 Command for interchanging adjacent words	
5.6.5 Command for pulling word down	

5.6.6 Command for removing junk characters	
5.6.7 Command for inserting certain words	
5.7 On Line Thesaurus .....	42
5.7.1 Extraction of Thesaurus	
5.7.2 Using the Thesaurus	
5.7.3 Enlarging the Thesaurus	
<b>6. Conclusion .....</b>	<b>45</b>
<b>Bibliography .....</b>	<b>48</b>
<b>Appendices</b>	
A. Roman Notation for Devanagari .....	i
B. Post-Editing Commands with Examples .....	iii
C. Some Standard Components of an Machine Translation System .....	xiii
D. Rules for Breaking and Performing <i>Sandhi</i> .....	xvii



# NOTATIONS USED IN THE THESIS

K	: Kannada sentence
T	: Telugu sentence
H	: Hindi sentence
E	: English sentence
@H	: Anusaaraka Hindi Sentence
!H	: Hindi Gloss
*	: An asterisk preceding a sentence indicates that the sentence is ungrammatical
abl.	: ablative
acc.	: accusative
dat.	: dative
emph.	: emphatic
erg.	: ergative
f.	: feminine
fut.	: future
instr.	: instrumental
m.	: masculine
nom.	: nominative
past_perf.	: past perfective
ppl.	: past participle
pres.	: present

# 1. Introduction

## 1.1 *Anusaaraka* Machine Translation Model

Translation is a creative process. It involves interpretation of a text by the translator. While coding information in a language, certain parts of the information become cumbersome to code and are not coded. Thus coding process sometimes loses information. Even in such a situation the reader is able to decode information using his background knowledge<sup>1</sup>. The act of filling the missing information by the reader using his background knowledge is called interpreting the text. Different persons may interpret the same text differently, depending upon their background knowledge.

It is a very difficult task to feed the background knowledge to the machine. On the other hand the machine is very powerful, in searching through a very large database (say a large dictionary of a few hundred thousand words) and, in following faithfully a given set of rules (say grammatical rules). The *anusaaraka* solution to machine translation is to cleanly divide the load between man and machine in such a way that language load is taken by the machine and the interpretation is left to the human. There are two principal points in this model where the user can intervene: pre-editing the input and post-editing the output. Besides this, the user should also be helped, in reading the output directly, and in learning the output language. In this there are issues regarding how to present the information in a controlled way to the reader, and to provide him help when he needs it.

This model is illustrated through figure 1.1. The typed in source language text is first pre-edited, and then it is given to *anusaaraka* as input. *Anusaaraka* produces output, which is in *amusarit* Hindi. This output can be read directly by a trained user, or it can be post-edited by a human post-editor and made available for a normal reader.

The user interact with the *anusaaraka* model at three different places, consequently the interface for *anusaaraka* has three subparts: pre-editing interface, post-editing

---

<sup>1</sup>general world knowledge or common sense knowledge, subject specific knowledge, knowledge of the context, etc. all these can collectively be called as background knowledge.

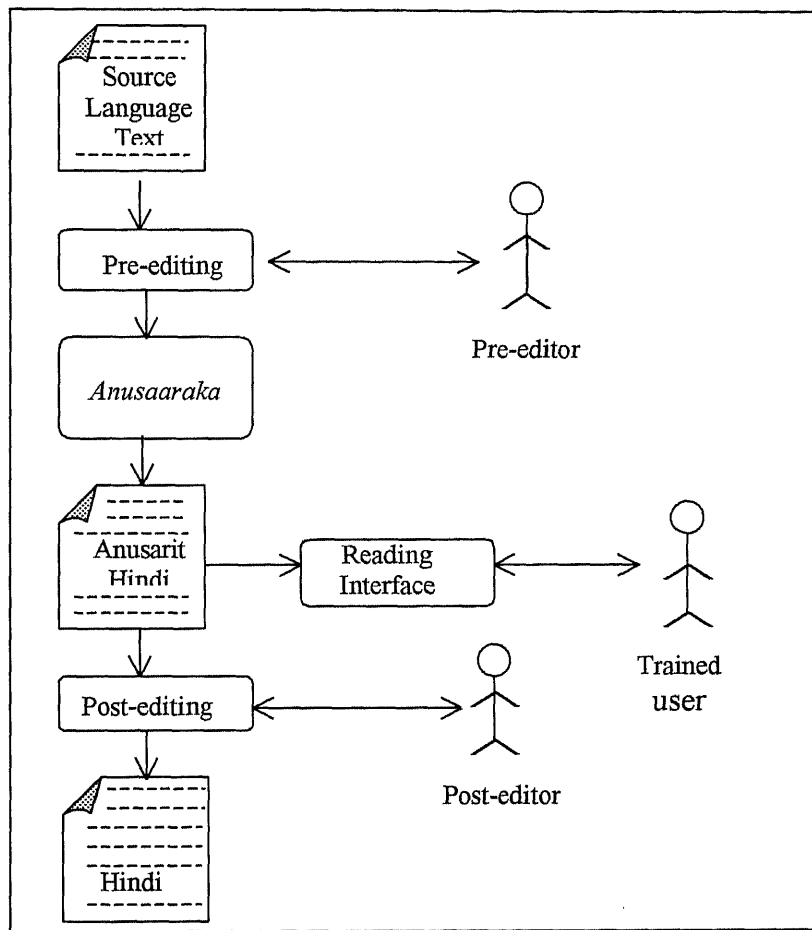


Fig. 1.1

interface and general reading interface. Now we will discuss the problems in these areas, and the motivation behind building these interfaces.

## 1.2 Pre-editing

A text before being given to the machine for running *anusaaraka* needs to be pre-edited. Pre-editing removes the following kind of errors in the text:

- **Correcting the typos:** Typing mistakes in a word will result in its being unrecognized by the machine. In pre-editing, all typos should be identified and corrected.
- **Standardizing spellings:** In Telugu a word is spelled in many different ways. Currently *anusaaraka* supports only the standard dialect of the source language. So the pre-editor will have to give the standard spelling for all the

words which are unrecognized due to the use of dialectal spellings.

- **Identifying proper nouns:** If the proper nouns are not to be translated in the output, then these proper nouns should be identified to the machine. The *amusaaraka* will then only analyze these words for *vibhakti* and present the proper output. Similar treatment has to be given to English words, which have found their way in the source language.
- **Breaking optional *sandhi*:** Optional *sandhi*, either done or not, doesn't change the meaning. The input should not have optional *sandhi*. Because for machine it is not very easy to break these *sandhis*.
- **Performing Compulsory *sandhi*:** Sometimes as a result of breaking *sandhi*, the meaning of resulting words may change. Such type of *sandhi* is called mandatory *sandhi*. If by mistake these *sandhis* are broken, they should be undone.
- **Correcting *sandhis*, which are broken at wrong places:** In Telugu sometimes the *sandhi* is done, but later on it is broken, but at a different place. E.g.

Word without *sandhi*:            manuSulu unnAru

After doing *sandhi*:            manuSulunnAru

After breaking *sandhi*:        manuSi lunnAru

So the word 'manuSulu unnAru' which has become 'manuSi lunnAru' will result in two unrecognized words for the machine. The pre-editor will have to correct these mistakes.

The machine can help the user in pre-editing by splitting or performing *sandhis*. Typically splitting or doing *sandhi* is governed by some rules. These rules affect vowels at the breakup points of the word in question. The machine can implement these heuristic rules and assist the user. The machine can also assist the pre-editor in telling standard spelling of dialectal words by maintaining a database, or by looking in the dictionary for the standard words with approximately same spelling.

## 1.3 Reading the output

The problem the user faces in reading the *anusaaraka* output is due to many phenomena of the source language, occurring in the same sentence of the output. The phenomena occur due to the difference in the source and target languages. In order to fill these differences some additional constructions are brought in, into the *anusaaraka* output language. All these phenomena in themselves might not be very difficult to learn, but sometimes they occur together in a sentence. Such a sentence becomes difficult to understand.

Some of the other issues are:

- Presence of all the information of the source text in the output. This information includes gender number person information, tense information etc. *Anusaaraka* brings all these information in the output, because it is faithful to the source language. However, they cause some unnaturalness in the output.
- Presence of words with back-quotes. These words represent the core meaning of the corresponding source language word. Their usage in the output language is not the same as in the target language. It also causes some problem to an untrained user.
- Presence of multiple options for a source language word.
- Presence of unfamiliar notation in the output. *Anusaaraka* uses some symbols like square braces, curly braces, etc. to represent some information. These are the symbols, which normally do not occur, in ordinary texts.
- Unknown words. Sometimes a reader may not know a word in the target language that appears in the output. This is a general problem a reader faces while reading any text. But here this problem by interacting with other difficulties becomes much greater problem and needs some attention.
- Failure of the machine in bringing out the target language equivalent word corresponding to certain source language word. This might happen because of following reasons:

1. Improper pre-editing of the text.
2. Incompleteness of dictionary and morph etc.

The basic objective of the reading interface is to help the reading of the output by the user. By the term 'helping the user' we mean:

1. The user should be able to understand the output. Here we will discuss two kinds of problem in understanding: mild catastrophe and serious catastrophe.
2. Increase the speed of reading.
3. Reduce the effort he has to put in, in reading, thereby making it a more pleasant experience.

Catastrophes[2] in *anusaaraka* output are said to occur when reader either fails to comprehend the meaning of a sentence, or misinterprets the meaning of the sentence. The former is called mild catastrophe while the latter is called serious catastrophes. The chief goal of the reading interface is to avoid these catastrophes. The user can read the output quickly and with less effort, if he is presented with appropriate amount of information in the first instance. If too many details are presented in the first instance, the output size will become huge and the user will have difficulty in understanding the meaning. If too little detail is presented, the user might be misled or will have to seek help repeatedly. Thus if the appropriate amount of information is presented to the user, he will be able to read the output quickly and without exerting much.

The reading interface is an attempt to address the above mentioned problems. It provides following features:

- **Controlling the amount of information presented to the user in the first output:** The guiding principle of *anusaaraka* is preservation of information. It means while producing the output, all the information present in the source text will be brought in the output. The interface may not present all the information present in the output, to the user at the same time. Some information is hidden, and is made available to the user when he asks for it. For example in case of multiple options it displays the best possible option within square brackets. It guesses the best out of the given alternatives on the basis of context using the context information database. If this guess is wrong

the user can select other alternatives and the context information database will be modified suitably.

- **Help regarding the words with back-quotes:** Words with back-quotes represent the nuclear sense of the source language word in the target language. In some sense it is a word which represents its core meaning. *Anusaaraka* uses this for information hiding, so that the number of alternative words that comes corresponding to the source language word is reduced to minimum. The interface provides brief help on these words telling all possible words whose meaning it represents. This help is displayed in a pop-up window, when asked by the user. For some of these words it also provides detailed tutorial.
- **Thesaurus help:** The synonym dictionary for the target language is embedded with the interface. The user can see all the synonym corresponding to a target language word, which is, unknown to him.
- **Providing help where machine failed in bringing out the equivalent target language word:** The failure of the machine in bringing out the equivalent target language words, may most of the time cause mild catastrophe. The interface provides dictionary lookup and *vibhakti* help for these.

## 1.4 Post-editing

Broadly speaking post-editing with respect to *anusaaraka*'s output has three levels. In the first level the output is made grammatically correct. The emphasis is on speed and on low cost, though output may still follow source language style. In the second level of post-editing, the raw output is corrected not only grammatically but also stylistically. All the artificial constructions are replaced by equivalent Hindi constructions. And in the third level of post-editing the post-editor might even change the setting and the events in the story to convey the same meaning to a reader who has a different cultural and social milieu. A creative post-editor can go all the way up to this level.

Here we will only concentrate on the first level of post-editing. In the following text by 'post-editing' we will mean the first level of post-editing. Typically in post-editing the user needs to do agreement, and many other grammatical corrections. These may involve adding *vibhakti*, selecting one word from multiple alternatives, replacing

artificial constructions by the target language constructions, filling information which was not there in the source text. These are more or less routine tasks. In the post-editing, the speed is very important. The role of post-editing interface is to speed-up the post-editing task, by automating all the routine tasks. It provides following features:

- **Support for agreement:** The user has to specify the noun and the verb that should agree with that noun in gender, number and person. Agreement is done automatically.
- **Support for adding *vibhakti*:** The user has to put cursor on the noun and invoke suitable command, as a result the *vibhakti* is added and any other adjustments are made automatically.
- **Selecting one word out of multiple alternatives:** For this there are two ways, first for person who knows the notation in which the alternatives are presented, and other for one who does not know the notation. The former is faster.
- **Replacing artificial construction:** One artificial construction 'jO\_\*' (for detail see chapter 2) is automatically changed to the corresponding proper Hindi construction, when the user supplies the missing *vibhakti* (which was not there in the source language and hence are not brought in the output).
- **Low level commands:** It provides miscellaneous commands for doing several things like converting a word into its plural form, interchanging two adjacent words, command for adding few commonly occurring words (which do not have their counterparts in the source language), command for removing junk characters etc.
- **On line thesaurus:** A user can replace a word by its more appropriate synonym from the synonym dictionary.

## 1.5 Objectives of the system

The issue is to design a practical system, which can be used in Indian environment. The major issues are that the system should be usable and cost effective. It should be within the reach of a common man. We preferred text-based tools for our system



rather than high-end graphic-based tools. Text-based systems are cheaper and can be made available to the public at lower cost. Further text-based systems are faster and also network friendly in the sense that they do not impose many loads on the network traffic by avoiding images.

An appropriate solution for our country is to provide the common man with 'Public Computer Centers' (just like public telephone booths). These centers may have ordinary 'Pentiums' (Rs 35000 each), with each Pentium connected to 4 Indian script terminals (Rs. 10,000 each). These computers will be running free software like linux and other gnu software). Thus the cost per seat will be only Rs. 15,000. Each terminal will support English and all Indian languages along with the Internet and email facility. The cost per person per hour will come around Rs. 5. With such kind of usable and affordable solutions, India can benefit more from the information revolution.

One disadvantage of using text-based tools is that the user has to remember commands, but that does not harm us anyway as we wish to train the user before he will use the system. The GUI consisting of menus, buttons and toolbox etc. is used by the user only initially. As soon as he becomes familiar with the system, he prefers to use accelerators. Accelerators are provided even in the GUI-based systems, so that the trained user can perform his work quickly.

In spite of all the obvious advantages that text-based tools have over graphic-based tools, the latter has importance. They are easy to use, intuitive to understand and are good for a beginner. They look appealing and cajole a beginner into using it. But these effects can also be brought in text based systems to some extent. It is for this reason we have emulated some such features in our text-based system and integrated it with our interface. We are using GIST technology for displaying multilingual texts. It is capable of displaying English and any other Indian language at the same time.

## **1.6 Issues in a good interface design**

In this section we will discuss the user interface tools employed in the *amusaaraka* interface. The knowledge about human factors plays a great role in designing quality interfaces. Designers often need to refer to design guidelines, to overcome their lack of human factor knowledge[3]. A good interface makes it easy for users to tell the computer what they want to do, for the computer to request information from users

and for the computers to present understandable information. Clear communication between the user and the computer is the working premise of good user-interface design[5]. Unfortunately all guidelines available in the literature are for high-cost graphic-based systems, with many fonts, many colors etc. Since ours is a text based system, they can not find their way into this tool. Some of the guidelines, which we discovered while designing this interface, are as follows.

- The interface should not ask the user to exert on his memory, For this we integrated drag and drop feature with our interface. The user has two windows, one has a set of tools, like a tool for running *amusaaraka*, tool for post-editing output file, tool for reading the output etc. And the other window has a list of file. So the user will select a file and click on the tool. The tool will be invoked with the selected file as input.
- The command name should be intuitive. It was found that if the name of the command could be derived from the work that the user is going to perform, then the user remembers that command and uses it quite often, rather than going for other alternatives. For instance in our post-editing interface the user has the whole text-editor at his disposal. He can do whatever he wants manually. If there is a command for doing it, and the name of the command is derivable from the task that he is going to do, then the command name will strike his memory and he will use it. For example in Telugu there is no counterpart for Hindi word 'Ora'<sup>1</sup> (aura), so this word is not present in the *amusaaraka* output. The user needs to type it often. In order to speed up this task the interface provides a low-level command. The user to do the work quickly can use this command. Typically The name of this command is 'ALT-O' in the 'vi' command mode.
- Standard tools like pop-up windows and pull-down menus are found useful for focussing the user's attention on a relevant part of the screen. Keeping this in mind they are emulated and integrated with the interface. Their use is made while displaying help and presenting multiple alternatives.

---

<sup>1</sup>internal representation for 'aura' in the machine.

## 1.7 Outline of the thesis

This thesis has six chapters. They are summarized as follows: the first chapter defines the problem this thesis is trying to solve. It discusses motivation behind *amusaaraka*, the general cost effective model suitable for Indian atmosphere, the tools and environment for supporting our Indian languages. The design of the interface is influenced by these parameters.

Chapter 2 discusses the general nature of the machine translation problem. The *amusaaraka* approach for handling this problem, *amusaaraka's* claim, and the detailed nature of the *amusaaraka* output languages, on top of which the interface is build.

Chapter 3 discusses the need for pre-editing a text before giving it to the machine for processing. It discusses the problem those pre-editor faces while pre-editing a text, and how machine can help him.

Chapter 4 discusses the solutions to the problems the user faces while reading the output. It describes the kinds of helps interface provides and how a user can use it. It also describes how the interface helps the user when he gets struck up.

Chapter 5 discusses various levels of post-editing, the kind of post-editing required in the output produced by *amusaaraka*, high level and low level tools provided by the interface to the post-editor for speeding up his task.

Chapter 6 concentrates on what has been achieved, and what can be done further, and how the post-editing interface can store world knowledge in the computer, side by side. We conclude by discussing future directions.

## 2. The *Anusaaraka*

### 2.1 Introduction

Fully automatic general-purpose high quality machine translation systems (FGH-MT) are extremely difficult to build. In fact, there is no system in the world for any pair of languages which qualifies to be called FGH-MT. The reason is that the translation is a creative process, which involves interpretation of a given text by the reader. While coding information in language, certain parts of the information become cumbersome to code and are not coded. While interpreting a text, a user fills up the uncoded information on the basis of his background knowledge<sup>1</sup>. Thus interpretation of a text might vary from person to person depending upon their background knowledge. A text is akin to picture made up of strokes and gaps. A viewer fills in the missing parts of the picture. If done properly, the reader gets the intended message.

### 2.2 The *Anusaaraka* Solution

The coding of background knowledge in a computer is a difficult task for machine. There are no known methods by which the machine can handle and use world knowledge today. Thus machine is weak in handling the world knowledge. But there are two aspects in which it is strong. It has a large memory, and it can perform arithmetic and logical operations very fast. For example, it can easily store a large dictionary of a few hundred thousand words, and it can search for a given word very quickly. Similarly, if the machine is given a grammar rule, it can apply it faithfully and with great speed. Much of language related data and rules could be fed into the machine, more easily than background knowledge.

#### 2.2.1 Sharing the Load

The *anusaaraka* solution is to share the load between man and machine so that the tasks, which are hard for the human being, are done by the machine and vice versa.

---

<sup>1</sup>general world knowledge or common sense knowledge, subject specific knowledge, knowledge of the context, etc. all these can collectively be called as background knowledge.

A clean way to share the load is for the machine to take up the task of language related processing, and to leave the processing related to background knowledge to the reader. Language related processing consists of analysis of the input source language text such as morphological processing, use of bilingual dictionary, and any other language related analysis or generation. These are the primary sources of difficulty to the reader. These are also the tasks, which are relatively easier for the machine. On the other hand, world knowledge related aspects are left to the reader, who is naturally adept at it.

### 2.2.2 Faithfulness vs. Naturalness

Faithfulness to the original text means whatever information it has, should be present in the output, whether it suits the target language style or not. Naturalness involves putting the output in the style and setting of the target language. Adding naturalness essentially means losing some information present in the original source language text and giving some new information. *Anusaaraka* prefers faithfulness to the naturalness. The *anusaaraka* output can be said to be the image of the source text, much like the camera image. Reading the image of the source text is like reading the original text. It will have the same flavor. Translation on the other hand, is like a painting. The translator interprets the original in the source language and "paints" a text in the target language in the same meaning.

There is a problem in coding "exactly" the same information (with 100% fidelity) from one language to another, particularly if we want to generate sentences of about equal length, paralleling the sentence constructions wherever possible. (In this sense, translation is sometimes said to be an impossible task). The *anusaaraka* answer lies in deviating from the target language in a systematic manner whenever necessary. This new language is something like a dialect of the target language.

## 2.3 *Anusaaraka* Output Language

The *anusaaraka* output language is not the target language, it is an image of the source language in a language close to the target language. Thus, the *Telugu-Hindi anusaaraka* produces a 'dialect' of *Hindi* that can be called as '*Telugu-anusaar-Hindi*'. It will contain some constructions of *Telugu* (which do not have equivalents), will not have gender number person (gnp) agreement, and will have certain artificial words, construction etc. In addition to this, it will also contain some additional

notation. Certain amount of training is needed for a user to get used to the *amusaaraka* output language. Because the Indian languages are close, the learning time of the output language is much shorter, and is expected to be around 2 weeks, as against the learning time required for learning the source language. Thus the training is worth the effort, because by a small effort one can read all Indian languages with the help of *amusaaraka*.

### 2.3.1 Artificial Words

*Anusaaraka* while producing output, replaces each word in the source language text by a substitutable word in the target language. A substitutable word corresponding to a source language word is one which when substituted for, in the output, triggers the right meaning in the mind of the reader. This should work in all possible contexts. Many times the substitutable word can be found because languages are close.

Usually a word has several meanings, but frequently they are all related to each other. If we imagine that each meaning or sense is a point in the sense space (or conceptual space), then the senses of the word would frequently be all clustered together. This cluster is called the sense space spanned by the word, or simply its sense space[2]. The nuclear sense of a word is a sense that, in a way, explains or gives rise to all the senses of the word. One can also define it as the sense that is equidistant from all other senses of the given word. Thus nuclear sense of a word has the property of substitutability as defined above.

*Anusaaraka* while producing the output, replaces each source language word, by a word in the target language that represents its nuclear sense. The target language word chosen to convey the nuclear sense has, in a way, different usage in the *amusaaraka* language, then in the actual target language. In the *amusaaraka* language its usage will be same as the usage of the source word in the source language. So when the target language reader reads the *output*, he is likely to be confused or even misled. Therefore these words are marked with 'back-quote', and some help or tutorial is provided for these words. For example if we compare *Bengali* and *Hindi* we find that they use the root '*kha*' for eating drinking as well as smoking. What will the substitutable word be for '*kha*' in Hindi. Obviously there is no exact word, therefore *amusaaraka* may use a new word, say '*kha*', and an online help will be provided by the interface for this word. These words marked with back-quote are in a way artificial words, but they are close to natural language words. This fact, helps the target user language reader learn

them quickly.

At this point some might argue that why not while producing output *anusaaraka* find out the object which is being eaten and accordingly put 'eating' or 'drinking' or 'smoking' whatsoever?

The answer is:

- *Anusaaraka* output is the image of source language. Some one might be interested in reading the original in the source language very closely rather than its translation. *Anusaaraka* is concerned with the demand of such a user.
- In the *anusaaraka* approach the machine translation is broken up into two modules[4]:
  1. The core *anusaaraka* system which is based on language knowledge and
  2. The domain specific module based on world knowledge, statistical knowledge, etc.

The things like "finding out the object being eaten and accordingly put the verb", comes in the domain of the second module.

### 2.3.2 Language bridges

Some problems arise due to difference in the two languages. There are only three major syntactic differences between Hindi and south Indian languages like Kannada and Telugu. Surprisingly all of these can be taken care of by enriching Hindi with a few additional functional particles or suffixes. These constructions will bridge the differences among the languages and the information will be carried across. These language bridges will be present in the *anusaaraka* output language in the form of additional constructions. In this section we will discuss these constructions.

#### 2.3.2.1 "ki" Construction

In case of embedded sentences in Hindi, the subordinate sentence is put after the main verb unlike in Kannada. For example:

H: rAma nE kaHA ki maiM ghara kO jAUMgA. (1)  
!E: Ram erg. said that I home acc. will\_go

E: (Ram said that he would go home.)

There is a construction in Kannada, which is similar (below, label 'K' stands for Kannada):

K: rAma hELiDanu EneMDare nAnu manege HOGuttEne. (2)

@H: rAma kaHA ki maiM ghara\_kO jAUMgA.

!E: Ram said that I home\_acc. will\_go

E: (Ram said that he would go home.)

However, it is seldom used. Kannada uses another construction for which the *anusaaraka* Hindi is given below.

K: mOHana nALe baruvanu eMdu rAma HELidanu. (3)

@H: mOHana kala AyEgA aisa rAma kaHA.

!E: Mohana tomorrow come-fut. that Rama said.

'aisa' construction is a proper construction in Hindi; only it is used less frequently. In the dialect of Hindi produced by *anusaaraka* from south Indian languages however, this will be the normal construction used.

### 2.3.2.2 "jO" construction

Another major difference between south Indian languages and Hindi is - South Indian languages are very rich in adjectival participle while Hindi has only two -viz. 'YA\_HuA', and 'tA\_HuA'. At the same time, adjectival participles in Hindi code the information of kaaraka relation whereas those in South Indian languages do not. Thus calls for new constructions in target language, so as to preserve the information.

Following examples, from Kannada, will make the point clear.

K : rAma tiMda cammacavannu toLe.

H : rAma khAyA HuA cammaca kO dhO DAlo.

The information regarding the relation of 'cammaca' to 'tinu(khA)' is not coded in the sentence. It is the background knowledge, which tells us cammaca is the instrument or theme of the verb makes the instrument for the verb 'eat (khA)'.

Now if we look at the Hindi sentences, the 'yA\_HuA' participle in Hindi codes karma(of sakarmaka verb and kartA of akarmaka verb).

E.g. khAyA\_HuA\_phala  
The eaten fruit.



Hence naturally the *anusaaraka* output of above Kannada sentences will sound odd for a Hindi speaker. Not only that he will 'read' something which is not there in the original, as uncoded information gets introduced.

Another problem is Kannada participles code information about tense, Aspect, Modality. Hindi participles however do not code this information.

Thus on the one hand, unnecessary information is getting coded, while on the other hand there is loss of information. This is because there is no one-to-one mapping between the participles in two languages and the information coded is also different.

To overcome this problem, it is necessary to identify a construction in Hindi, which codes the information in exactly same way as the South Indian Languages.

The 'jO' construction in Hindi is the answer for above problem. Look at the sentence

H : rAma nE khAyA thA jisa cammaca sE usako dhO DAlo.

One can rewrite the above sentence, without any loss of information, as

rAma nE khAyA thA jisase usa cammaca ko dhO DAlo.

similarly, one can have

rAma nE banAyA thA jisako usa cammaca ko dhO DAlo.

rAma nE (cAvala) banAyE thE jisameM usa bartana ko dhO DAlo.

and so on. This construction also explicitly codes the information about kAraka relation. Now consider the following sentences obtained from above sentences but which do not have any information about the kAraka relations.

rAma nE khAyA thA jO\_\* usa cammaca ko dhO DAlo.

rAma nE banAyA thA jO\_\* usa cammaca ko dhO DAlo'.

rAma nE (cAvala) banAyE thE jO\_\* usa bartana ko dhO DAlo.

These sentences code the exact information as is coded by corresponding Kannada sentences. The '\*' indicates appropriate post-position marker whichever is applicable in the given context is to be supplied (by a human reader)

### 2.3.2.3 'nE' construction

The "nE" construction or ergative marker is a peculiarity of only the Western belt

languages in India. In case of the present or past perfect aspect of the main verb in Hindi sentence, "nE" is used with the karta:

H: rAma nE phala khAyA.  
!E: Ram erg. fruit ate.  
(Ram ate the fruit.)

In *anusaaraka* output from Kannada to Hindi, the 'nE' post-position would never be produced. It would not be produced even with the TAM label 'yA' in Hindi (wherein it is mandatory barring a few exceptions verbs). For example:

H: rAma phala khAyA.

Therefore, we can postulate a new TAM (yA`) with same semantics as "yA", but which does not use "nE" construction in *anusaaraka* Hindi. With this TAM, we can express the corresponding Kannada sentence more faithfully as:

@H: rAma phala khAyA`.

### 2.3.3 Agreement

Let us consider the case of noun-verb agreement. There is a lack of agreement (of gender, number and person) as per the rules of the target language in the *anusaaraka* output. The information about gender etc. is displayed corresponding to the source language. For example, in the *anusaaraka* output below, {m.} and {f.} respectively against the personal pronoun 'vaHa' mark the masculine and feminine gender. ({~m.} stands for non-masculine). Note that in Hindi, personal pronoun 'vaHa' is the same for both masculine and feminine gender.

T: Ame VADito mATLADiMdi kAnI,

@H: vaHa{f.} usa{m.}\_se bAta\_kiyA\_[Hai/thA]{3\_~m.\_e.}  
lakina{Hone\_dO},

!E: she he{instr.} talked{non\_masc.} but,

T: vADu Ameto mATLADaledu.

@H: vaHa{m.} usa{f.}\_se  
bAta\_kiyA\_naHIM[naHIM\_bAta\_kara\_sakata\_Hai{3\_~m.\_e.}].

!E: he she{instr.} did[could]\_not\_talk{non\_musc.}.

E: She talked to him, but he did not talk to her.

If the gender information were not shown, the sentence rendered would have been rendered as:

H: `usanE usasE bAta kI, lekina usanE usasE bAta naHIM kI.`

!E: `s/he s/he{instr.} talked, but s/he s/he{instr.} talk not do.`

Without the gender information in *amusaaraka* Hindi, the meaning of the sentence is not clear. To produce good Hindi from such a sentence, requires different strategies. One solution would be to explicitly add 'laDakA' (boy) etc. indicating the sex:

H: `usa laDakI nE usasE bAta kI, lEkina usa laDakE nE usasE bAta naHIM kI.`

!E: `that girl-erg. her/him talked, but that boy erg. her/him talked not.`

But whether it should be boy, or man or something else would depend on the context, and quite beyond the capacity of the machine to infer correctly in all possible situations. Another solution would be to change the tense-aspect label slightly so that it becomes different from post-complative (at the cost of faithful to the original). By doing this, 'karta-verb' agreement would no longer be blocked by the post-position marker and show the gender in the verb. Yet another solution would be to use 'bolatA\_Hai' (speaking) a construction in which agreement between noun-verb specifies the gender of the karta of the speaker.

H: `vaHa usasE bolI, per vaHa usasE naHI bolA.`

!E: `spoke{f.} spoke{m.}`

Appropriate selection and use of such strategies is left to the post-editor in the *amusaaraka* approach.

### 2.3.4 Multiple Options

*Amusaaraka* will perform the word sense disambiguation only if it can be performed based on language knowledge. Such disambiguation will be robust and will never fail.

It does not perform the disambiguation, which is based on world knowledge. The reason for this is: disambiguation based on world knowledge is difficult for the machine and is also error prone. And *anusaaraka* wishes to present a faithful image of the source language avoiding any source of errors at its level. On the other hand, doing word disambiguation is very easy task for humans. For disambiguation, world knowledge as well as context knowledge is required. So in *anusaaraka* output all the senses of a word are given, and the user is supposed to disambiguate them on the basis of world knowledge and the context.

In case translation task is in a limited subject domain, such sense disambiguation can possibly be done reasonably well. In the *anusaaraka* view, such a disambiguation (and use of world knowledge in general) can and should be done after the *anusaaraka* stage.

### 2.3.4 Extra Information

Different languages code different information differently and at different places. Certain information coded in the source language at one place might be coded in the target language at some other place. So the information coded in the source language may not be needed by the target language at that particular place. Such information is presented in the output within the curly braces, attached to the word to which it pertains. Some times such information is very important in grasping the meaning, as is evident from the following example:

```
nEnu monna   HaidrAbAdulo   unnAnu.
maiM  parasoM  ^^^           Hai/thA/raHA_[Hai/thA].
      {bItA_HuA}
```

Different languages code different information at different places. For example, in *Telugu* the information '{bItA\_HuA}' is encoded in the word 'monna', whereas in *Hindi* the same information will be coded in 'Hai' or 'thA'. Since 'Hai' or 'thA' does not have their counter parts in *Telugu*, extra information has to be presented, otherwise there will be information loss.

In the above example 'parasoM' of Hindi is ambiguous, and the extra information within curly braces helps the reader in disambiguating it, without which the user might have misunderstand it. Such extra information is also present regarding gender, number and person of words. Many times these information are used and absorbed by

the post-editing command tools in the post-editing interface. Sometimes however, the extra information may be redundant, but it does not pose any problem.

## 2.4 The *Anusaaraka* Claim

The text in any language, when read by a reader of that language, conveys some information to the reader. The reader interprets it on the basis of, the actually coded information, his world knowledge, and his cultural and social background. If the same text is run through *anusaaraka*, and is subsequently read by a reader who knows *anusaaraka* language, then precisely the same information will be conveyed to him. Provided both readers have same world knowledge, and social and cultural background. They will interpret it in similar ways. (Fig. 2.1)

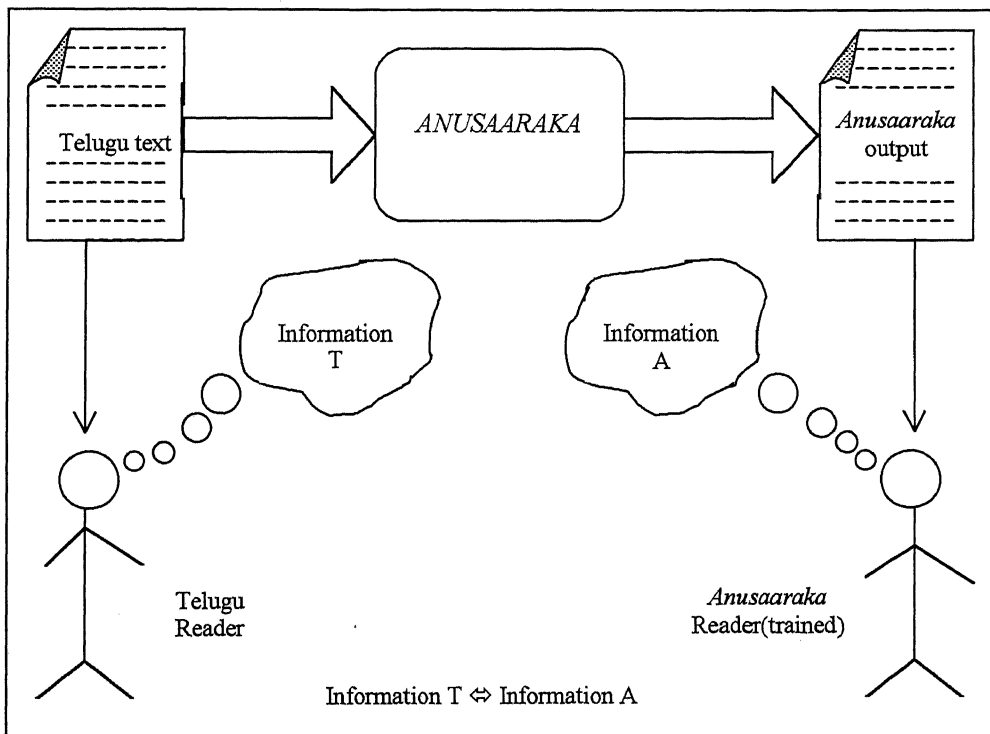


Fig. 2.1

# 3. The Pre-Editing Interface

## 3.1 Introduction

This thesis designs and develops an interface for *Anusaaraka*. The interface has three subparts: the pre-editing interface, the general reading interface and the post-editing interface.

In the pre-editing task, the input text is corrected and edited by the user: Word spelled with non-standard spellings are changed to their standard spellings, external *sandhi* between words is broken (unless it changes meaning), proper nouns and foreign words are identified etc.

This is an important task for Indian Languages because of the lack of standardization and consequent variation. It is particularly serious in *Telugu*, where the spelling variation is very large. On an average a word can be written in three alternate ways. Written *Telugu* allows considerable amount of an admixture of local dialects. Partly the reason for this is that the written material has been influenced by the local dialects in the last forty years. In fact, young and influential writers actively promoted use of local dialects of *Telugu* in written texts in this period. There has also been no major effort at standardization.

Similarly, there is lack of standards in the use of spaces. Sometimes *sandhi* between words is performed, sometimes not so. Worse still for the machine, when *sandhi* results in a long word, it is broken up at a point different from where the *sandhi* has been done. The machine will thus have difficulty with both the resulting words.

Spelling variation might be severe in written *Telugu*, but is present in all Indian Languages. Much more so than say in English. One will have to live with this reality, while designing a machine translation system.

## 3.2 Why Standardization in a language?

There are many reasons for why a language should be standardized.

- Standardization is a way through which deviations in the languages can be

controlled.

- Standardization facilitates reading and comprehending text by a large number of readers. Standardization helps in learning as well as using the language.

### 3.3 How Standardization helps *anusaaraka*

Some people might argue that the machine must handle all the variations. For the phenomena mentioned above, in principle, it does not seem to be a problem. However, in practice, it requires a much bigger effort. Instead of three years it might mean twenty years of efforts to develop a working system. So it has been decided to build the system only for a sub-language first, say, the standard language (to the extent defined already, or by extending the definition) Later on this system can be extended to cover many dialects, thereby reducing the task of pre-editing. However the important point is that the sub-language is so chosen that sufficient amount of written material exists which is needed by other language groups or persons, for the system to be useful.

Since *anusaaraka* supports only standard dialects, pre-editing of a text is required before *anusaaraka* can run on it. The objectives of pre-editing interface are:

- **Correcting the typos:** The machine may not recognize a word because of typing mistakes in it. For example in following word a space is typed by mistake:

bAbi gADito

Therefore they have become two unknown words for the machine. In pre-editing these two words should be combined as 'bAbigADito' (meaning 'bacchE\_sE').

- **Enforcing standard dialect:** Language should have standard dialect. But in Telugu many dialectal spellings for a word are used quite often. For example the Telugu word 'seVlavu' (in standard dialect) is also spelled as following variants.

shalavu, selava, shalava (meaning is 'chuttI' or leave)

In stead of using so many variants, the single standard spelling should be used.

The machine faces problem due to these spelling variations. It is the job of the pre-editor to replace these dialectal words by their standard spellings. The machine can help the pre-editor by looking into the dictionary and finding out the words with approximately same spellings, asking the user to select the appropriate one. This however has not proved to be a good strategy in practice, as often the list of words brought out by the machine is very long. The interface does not have much help to offer in this case. It however opens a new area of research for linguists. This area will involve finding the logic behind spelling variations, and finding rules, if there are any, to automate it.

- **Splitting optional *sandhi*:** Optional *sandhi* means performing *sandhi*, which does not changes the overall meaning of the sentence. For example.

mEku UDiMdi            or mEkUDiMdi (meaning kIla nikAlI)

iMTiki veLLi or iMTikeLLi (meaning ghara jA\_kara)

paMDu tini            or paMDutini (meaning phala khA\_kara)

Here it is necessary to mention that if two words are written together without 'space' between them, then also it will be called as *sandhi*. Because for machine breaking up such words will be same as breaking *sandhi*, in terms of effort. Doing optional *sandhi* is not a good practice. Modern Indian language has a convention of avoiding it. This also makes the task simpler for the machine.

The task of breaking *sandhi* is difficult for the machine, and is also error-prone. On the other hand, it is easy for the pre-editor and so is left for him. The interface provides command for performing *sandhi*. Breaking *sandhi* affects the vowels at the break points. There are some heuristic rules (see appendix for rules for performing *sandhi*), which determine these changes. So the interface tries all applicable rules, and presents the results to the user in a pop-up window, the pre-editor can select the appropriate one.

- **Mandatory *sandhi*:** Sometimes as a result of breaking *sandhi*, the meaning of resulting words may change. Such type of *sandhi* is called as mandatory *sandhi*. There is no choice regarding either to break it or not. It has to be left as it is. The pre-editor should not break such *sandhis*. On the contrary if, by



mistake such *sandhis* are split in the text, they should be undone. The interface also provides commands for helping the reader in this aspect. After the user puts the cursor at the unrecognized word and invokes a suitable command, the interface picks up that word and analyzes it for breaking. It then breaks the word if any of the given rules (see appendix for *sandhi* breaking rules) are applicable. After breaking the word, it checks if resulting words are acceptable to the machine. If the resulting words are acceptable, they replace the original word, if there are more than one such words, they are presented to the user in a pop-up menu, for selection.

- **Identifying Proper Nouns and Foreign words:** Machine is unable in identifying proper nouns in a given sentence. If there is entry corresponding to the proper noun in the dictionary, then it will be analyzed and translated producing incorrect result, otherwise it will be produced in the output as an unrecognized word. If the proper nouns are to be brought as they are, they need to be marked by the post-editor. *Anusaaraka* has simple notation for marking them. For example the proper noun 'rAmuni' will be marked as ': :rAmuni' in the input text. The machine will only analyze these for *vibhakti*, and output them as it is, along with the *vibhakti*. For example, ': :rAmuni' will be brought out as 'rAma\_kA'

Similarly the English words, which are found often in any text, must also be marked, so that the machine can identify them. *Anusaaraka* assumes a word to be an English word or any other foreign language word, if it starts with a colon ':'. These are given similar treatment as proper nouns. These are not translated. However the machine analyzes them for *vibhakti*. For example, the word ':klabbluO' will be outputted as 'klaba\_mE'.

### 3.4 Using the Pre-Editing Interface

The first step for doing the pre-editing is to identify the words, which are unrecognized by the machine. The interface provides a command, which when the user invokes, marks all the unrecognized (to the machine) words in the current line. The user then takes the cursor on the next unrecognized word. (The interface provides commands for taking the cursor on the next, or on the previous word, 'TAB' and 'Shift-TAB' respectively.) Then the user will have to make a guess, regarding the type

of the action to be taken. If there is a typo, the user will have to be correct it manually. After correcting it, he will verify its validity using the same command again. If it is a proper noun, or foreign word, the user will have to mark it accordingly. If a *sandhi* has to be done or split, the user will invoke the appropriate command, as a result the word will be broken or combined (with its adjacent word). If breaking or combining results in more than one choice, the choices will be presented to the user, for selection.

CENTRAL LIBRARY  
I. I. T. KANPUR

Acc. No. A 125386

## 4. The General Reading Interface

The *amusaaraka* output is in a language, which is not target language, but is close to it. So person who knows the target language might face problems in reading it. The aim of reading interface is to help the reader while reading the output. It tries to present appropriate amount of information to the reader in the first instance. It also provides on line help. The online help includes help on special symbols used by *amusaaraka* to convey information, help and tutorial on artificial words used in the output, help on unrecognized words etc. In this chapter, we will discuss the solutions provided by the post-editing interface to help the reader, in greater detail.

### 4.1 Controlling amount of information

It has already been discussed that the *amusaaraka* produces output trying to preserve the information in the source text. At the same time it is important that the information is presented to the reader in such a way that he is not inundated with it. One objective of the reading interface is to present the output in a readable form, by controlling the amount of information presented. For example certain details which are likely to be less important initially are hidden from the user.

One reason for the large amount of information is in the output that there might be many possibilities for a source language word in the target language. Even though great care is taken in controlling the number of alternatives while making the dictionary (see nuclear sense etc.), there might still be many alternatives for some words. For example, if three of the content words have two meanings each, it would generate eight possible readings. The general suggestion for the reader is to first go through the output once quickly, avoiding the alternatives in the first reading, unless absolutely necessary. In order to facilitate this suggestion the interface hides all the alternatives corresponding to a word, and shows the best possible alternative within square braces. It decides the best alternative on the basis of the context. It keeps context information for some words in a database. Initially the database is empty, and it will assume that the first alternative is the best. This is justified because *amusaaraka* tries to bring the most frequently used meaning first. It will also save the preceding and the proceeding word of the first alternative as its context. If, however the user

finds that the alternative is not suitable, he can choose the other one, and the context information database will be modified accordingly. For selecting other alternatives, the user can press a command, after putting the cursor on the word and a pull-down menu appears displaying all the other options. Thus as the interface will be used more and more, its context-information database will keep getting matured.

Example:

```
T : vADiki pAtika vEla appu I rOju tirigi ivvAli.
@H: usakO`{pu.} chauthA_bhAga/paccIsa HajAra karja yaHa-
dina/HANpha lauTA_[kara]/phira_sE dEnA_cAHiE.
@I: usakO`{pu.} [chauthA_bhAga] HajAra karja yaHa- [dina]
[lauTA]dEnA_cAHiE.

H : usakO paccIsa HajAra kA karja is dina lauTA
dEnA_cAHiE.
```

In the above example by controlling the amount of information, the interface has made so complex looking sentence so simple. The user may feel that the second word has problem. Just clicking on it will give following result.

```
@I:[usakA] 

|                   |
|-------------------|
| 1.[chauthA_bhAga] |
| 2.[paccIsa]       |

 HajAra karja yaHa- [dina]
[lauTA] dEnA_cAHiE.
```

Selecting the correct choice will yield the following.

```
@I:[usakA] [paccIsa] HajAra karja yaHa- [dina] [lauTA]
dEnA_cAHiE.
```

Initially it had saved the context of 'cauthA\_bhAga' as a word preceded by 'usakA', and followed by 'HajAra'. Later on, after the user makes selection, the interface deletes the context information regarding 'cauthA\_bhAga' and saves it

as a context information for 'HajAra'.

## 4.2 On Line Help

The reading interface also provides the user with online context-sensitive help. It provides help regarding the notations used in the output, and brief help as well as detailed tutorials on the artificial words introduced in the output.

### 4.2.1 Help on Notation

The *anusaaraka* uses many symbols like bracket, back-quote etc. in the output to convey information. The reader may face problem in reading the output because he does not know or remember the meaning of certain symbol. To aid the user the interface has on line help. To invoke help the reader has to put the cursor on the symbol and execute the help command. The interface picks up symbol under the cursor and displays help corresponding to it in a pop up window. User can also see help regarding all the notations used in a word. In this case the interface will pick up the word under cursor, analyze it for whatever helps is available on all the symbols used in it, and displays the help in a pop-up window. For example, let us consider the following sentence:

T: O prakhyAta nAyakuniki gaurava sUcakaMgA O kAMshya  
+VigraHaM cEyiMci nilabeTTAlani baMdaru paurulu  
utsAHapaDDAru.

@H: Eka prakhyAta- nAyaka\_kO`{pu.} gaurava- sUcaka\_\* Eka  
kAMshya + vigraHa karA\_[kara] khaDA\_karanA\_cAHiE\_aisA  
machalIpaTTaNam[<baMdaraGHa]  
nAgarika{ba.} utsAHita\_HuA\_[Hai/thA]{23\_ba.}.

H: Eka prakhyAta nAyaka kI Eka gaurava sUcaka kAMshya  
mUrTi kaDI karanI  
cAHiE, aisa sOch kara machalIpaTTaNam kE nAgarika  
utsAHita\_HuA.

In the above example the *anusaaraka* has used many notations, if a user wants to see

help on any of these, he can invoke help command by putting cursor on that symbol. For example putting cursor on '<' will yield following help.

\_\_\_\_\_madada\_\_\_\_\_

machalIpaTTaNaM[<baMdaragAHa] : bIja artha 'baMdaragAHa'  
 kintu 'machalIpaTTaNaM' bhI saHAYaka HO sakatA Hai.

\_\_\_\_\_iti\_\_\_\_\_

#### 4.2.2 Help on Words with 'Back-quote'

As already mentioned, some words in *amusaaraka* output are marked with 'back-quote'. These are words of the target language, which represent nuclear sense for source language words. The usage of these words in the output language is slightly different from their usage in the target language. So interface provides brief help regarding these. The help can be read by putting cursor on the corresponding back-quote and executing the suitable command. For example, let us consider the following sentence:

T: oVka klabbulo EdO                      utsavaM jarugudoMdi.  
 @H:                      Eka                      klaba\_mE                      kOI[kucha]utsava  
 saraka`\_raHA\_[Hai/thA].

In the above example, the word 'saraka' has 'back-quote' marked on it, the user can see help on it by putting the cursor on the 'back-quote' and invoking help command. The following help will be displayed:

\_\_\_\_\_madada\_\_\_\_\_

'saraka`' - sarakanA, HOnA, bItanA ..

After reading this help, the user can make out the correct Hindi sentence as:

H: Eka klabamE kOI utsava HO\_rAhA\_thA.

### 4.2.3 Tutorial on Words

Besides the brief help, the interface also provides detailed tutorials on some of the words. Such words are enclosed in "(e.g. 'kO')<sup>1</sup>". When user invoke suitable command after putting cursor on such words, the interface looks for a specific file in some specific directory. This file contains the name of the tutorial file corresponding to this word. It then opens that tutorial file in the same window. The user after reading the tutorial can revert back to the original file.

### 4.2.3 Thesaurus Help

Sometimes the difficulty in understanding the output arises because the reader does not know the target language word that appears in the output. To help the reader in such situations, thesaurus help is embedded with the interface. The user can look into the synonym dictionary corresponding to the target language word and read its synonyms and select the appropriate one. Typically for viewing the thesaurus help, the user will put the cursor on the target language word, and invoke the appropriate command. The synonyms will be shown in a pop-up menu. For example consider following sentence:

T : A                      manisi                      lOpaliki                      pOvaDAniki  
siddhapaDDADu.  
@H: vaHa- manuSya aMdara\_kO jA\_nE\_kO`  
sannaddha\_HuA\_[Hai/thA].

In this sentence the user may face difficulty in reading the output because of the word 'sannaddha'. He may seek thesaurus help. After placing cursor on the word and evoking thesaurus help will present following:

---

<sup>1</sup>This is standard 'tags' notation of 'VIM' (VI iMproved, distributed with linux). To invoke such help the user has to run the same command as for viewing 'tags' information in 'VIM'

vaHa- manuSyA aMdara\_kO jA\_nE\_kO  
\_HuyA\_[Hai/thA].

- 1.sannaddha
- 2.taiyAra
- 3.udyata
- 4.pUrA

After selecting the easier synonym the by the user sentence will become:

vaHa- manuSyA aMdara\_kO jA\_nE\_kO taiyAra\_HuyA\_[Hai/thA].

## 4.3 Unrecognized Words

The reading interface also provides help on the words, which are not recognized by the machine. Sometimes they are crucial for the reader to grasp the meaning of the text. A word might not be recognized by the machine because either

1. The machine is not able to analyze its form (though its the root is in the dictionary), or because
2. The root is not in the dictionary (even though morph is capable of analyzing the word form).

If only one of the above conditions holds, the machine can help the reader in guessing or arriving at its meaning. Typically the help on unknown words tells the *vibhakti* of the unknown word, or picks-up words from the dictionary, which have approximately the same spelling. The unrecognized words fall under two categories.

### 4.3.1 Foreign Words

These are words taken from other languages, mostly from English. For example, consider a word commonly used in *Telugu* 'steSanlO'. This word will come as it is, in the output since the word 'steSan' (station) is not likely to be in the *Telugu-Hindi* dictionary. The interface provides *vibhakti* help in such cases. The interface analyzes the word under cursor for *vibhakti* and presents it to the user. When the user knows that 'lO' is 'mEM', he can guess the meaning of the remaining



(steSanlO). Often *vibhakti* help is sufficient in case of foreign words<sup>1</sup>. The *vibhakti* is also helpful in case of proper nouns, which comes as they are, in the output.

### 4.3.2 Unrecognized Words due to incomplete Morph

The problem of unrecognized words of this kind will remain for a long time, for languages with a rich morphology such as Telugu, Kannada etc. So some help will always be needed on these. Interface provides help on these words by showing the dictionary entries with approximately the same spelling. Hopefully the approximate match will yield the root of the word, and the reader will be able to get the approximate meaning of the word. This help is quite useful, but is rather slow. It has been provided using Unix 'agrep' utility.

The problem regarding unrecognized words is similar to what any reader of a normal text faces as it seems. In fact when we read some text, we do not know the meaning of all the words in that text, nor do we consult dictionary for each unknown word, still we are able to understand the meaning. Similarly if some words are unknown in the output, their meaning can be guessed by the context. However, what makes the problem more serious here is that there might be many other difficulties (lack of familiarity of the reader with *amusaaraka* language, incompleteness of the system, lack of standardization in spellings etc.) which together combine to make the output text harder to understand. Therefore *amusaaraka* lays great emphasis on the coverage of the words (above 95%) and in addition provides help through the interface just described.

---

<sup>1</sup>However the reader still has to learn the spelling style of the foreign word in the source language

# 5. The Post-Editing Interface

## 5.1 Introduction

It has already been discussed that the *anusaaraka* output is close to the target language, and in general is not grammatical from the target language viewpoint. In case, a user is reading for his own sake, he might not bother to produce a grammatically correct and stylistically more suitable output. However, when a document is going to be distributed in large numbers, it would normally be post-edited by a person before distribution or publication.

There are three levels of post-editing. the first level of post-editing seeks to make the output grammatically correct. The emphasis is on speed and low cost. The post-editor might drop phrases, change construction in the interest of speed, as long as it does not alter the gross meaning. Under this level of post-editing, corrections are made regarding agreement, putting 'nE' (ergative marker) where necessary (see Section 2.3.2.3), inserting the correct *vibhakti* in 'jO\_\*' construction (see Section 2.3.2.2), etc.

In the second level of post-editing the raw output is corrected not only grammatically but also stylistically. There can be many different types and quality of output at this level, depending on the audience. One audience might be willing to accept some constructions in the raw output which are grammatically correct in Hindi but not used often. Another audience might not be willing to accept it. For example, 'aisA' construction (see Section 2.3.2.1) can be changed to 'ki' construction, for such an audience.

In the third level of post-editing the post-editor might change the setting and the events in the story to convey the same meaning to the reader who has a different cultural and social milieu. This is really Trans-creation, and a creative post-editor can go all the way up to this level.

A post-editing interface allows him to do post-editing rapidly. Rather than making corrections character by character, he can supply the missing information and the

feminine plural, he need not individually change the verb and its auxiliaries to the correct forms manually. Instead, he can place the cursor on the verb sequence and give a command, and the computer would change the forms of the verb and its auxiliaries.

## 5.2 Gender Number Person Agreement

Hindi has an agreement rule which can be stated as follows:

Gender, number and person (gnp) of the verb agrees with the gnp of the *karta* if it has 0 *vibhakti*; otherwise the gnp of the verb agrees with the *karma* if it has 0 *vibhakti*; otherwise the verb takes masculine singular third person form[2].

The agreement in the *anusaaraka* output follows the rules of the source language. This apparent loss of agreement is not expected to be too jarring to Hindi speakers in view of their exposure to various varieties of non-native Hindi heard everyday, as propagated by television, radio and films. Still it has been decided to include the gender number person agreement in the human post-edited output. Since it is difficult for the machine to do it automatically(see sec 2.3.3), the task is left to the post-editor. This gnp agreement can be done by the post-editor manually using any editor, but in order to increase the post-editing speed it is necessary to provide tools for doing it.

In the tools provided, the post-editor has to select the verb and the noun (or pronoun) with which it's agreement has to be done, and the agreement is done automatically. Let us consider the following example:

```
maiM{pu.} ghara_kO` jA_raHA_Hai.
```

In the first step the user puts the cursor on the pronoun 'maiM' and invoke a suitable command. As a result the interface looks for its gender, number and person. These should be available either in the database or in the output itself. If these are not found from either, then the user will have to give the missing information using some simple commands. In the above example the interface will find the number and person information from the database, and the gender information from the output itself. In second or final step the user puts the cursor on the verb 'jA\_raHA\_Hai' and invokes another command. Consequently the interface looks into the intermediate

file<sup>1</sup> for the root and the TAM<sup>2</sup> of the verb. All these information will be given to the Hindi generator which will generate the right form of verb. The resulting sentence will be:

```
maiM ghara_kO` jA_raHA_HUN.
```

In most of the cases the verb, subject to agreement is the last verb of the sentence, So in order to automate it further, if the user does not specify the verb, the program checks every word starting from end of the sentence and performs agreement on the last verb of the sentence. For example:

```
<s1> maiM{pu.} ghara_kO` jA_raHA_Hai.<.>
```

Here the user is required to keep cursor on 'maiM' and invoke suitable command and the last verb, i.e. 'jA\_raHA\_Hai' will be made to agree accordingly. This works only if the output is marked with end of sentence marker (<.>). By default, in the *anusaaraka* output the end of sentences are not marked, however the user can do it, using certain commands.

## 5.3 Noun Phrase Agreement

The noun-phrase agreement or adjective grouping is also dropped from the *anusaaraka* output. What noun-phrase agreement means, can be explained by following example:

```
kAlA mOTA laDakA.
```

Here 'kAlA' and 'mOTA' are adjectives and 'laDakA' is noun, together 'kAlA mOTA laDakA' becomes a noun phrase. Peculiarity with noun phrase is that, if the form of noun changes, the form of related adjectives will also need to be adjusted. For example if 'laDakA' becomes plural then the noun-phrase will become 'kAlE mOTE laDakE'.

### 5.3.1 Adding *Vibhakti*

Typically the 'nE' post-position or ergative marker is not present in Telugu so they

---

<sup>1</sup>intermediate file is generated by *anusaaraka* while producing the output, this file contains the intermediate data produced, while processing the input file.

<sup>2</sup>Tense Aspect Modality.

are not present in the output as well. These are to be put by the post-editor. Addition of the *vibhakti* affects the forms of the noun and the adjectives in the noun phrase.

For example let us consider following sentence as produced by *anusaaraka*.

kAlA mOTA laDakA phala khAyA.

After putting the 'nE' post-position after 'laDakA' the sentence will become 'kAlE mOTE laDakE\_nE phala khAyA.'. Please notice that the whole noun phrase is altered as a result of adding the 'nE' post-position marker. Interface has a command for automating it. In this the user has to specify the boundary of the noun phrase. For example in the above example, the user will place the cursor at the word 'kAlA' and invoke a suitable command. As a result the interface will mark 'kAlA' as the boundary of the noun phrase. Next the user will take cursor at the word 'laDakA' and invoke command for adding 'nE' post-position. As a result the sentence will be modified as:

kAlE mOTE laDakE\_nE phala khAyA

In order to further automate it *Anusaaraka* has a special notation for identifying the attachment of adjective with noun: such adjectives end with an underscore('\_'). Thus when the user does not specify the boundary, the interface automatically looks for preceding words. It keeps on looking as long as it finds words marked with underscore ('\_'). For example if the above sentence brought out as following by *anusaaraka*:

kAlA\_ mOTA\_ laDakA phAla khAyA.

Invoking the noun-phrase agreement command by putting the cursor on 'laDakA' will automatically adjust the adjective groups. The above sentence after using the mentioned tool will become:

kAlE\_ mOTE\_ laDakE\_nE phala khAyA.

The second alternative is of course more preferable, but some times due to some error somewhere *anusaaraka* is unable to mark the adjectives for attachment (with the following noun). In such cases, first alternative will be used.

Similar support for inserting other post-position markers like 'kA', 'kO', 'kE', 'kI', 'maiM', 'sE', 'para' etc. is also provided. This will be

needed when these post-position markers are not present in the output. This may not be present in the output either due to mistake in morph function or dictionary, or because the source language has an absence of *vibhakti*, whereas the target language does have a *vibhakti*.

### 5.3.2 Giving Gender

In the above example since the gender information is not present, the interface is assuming the default gender 'male'. Many times the gender of noun is not there in the output, and is to be given by the user from the context. For example 'kutta' (dog) in Telugu is neuter gender, but in Hindi it has either male or female gender. So the information regarding the gender of 'kutta' will not be there in the *anusaaraka* output. Post-editor has to guess it from the context and give it before invoking the command for adding *vibhakti*. For example consider sentence.

```
kAlA_ mOTA_ kutta baccA diyA.
```

After giving gender information (using command for adding female gender) and subsequently invoking command for adding 'nE' post-position marker will result in following:

```
kAlA_ mOTA_ kutta{strI.} baccA diyA.
```

```
kAlI_ mOTI_ kutiA_nE baccA diyA.
```

## 5.4 Selecting from Multiple options

For the reason already mentioned *anusaaraka* does not perform word sense disambiguation. The consequence is that for some words there will be multiple meanings, and the user will have to disambiguate and select the appropriate one. The multiple options are stated briefly in rather precise form so that they take minimum space on the screen. Before going into the interface part, first we will discuss the notation.

Alternatives involving two or more words can be presented by using '/' For Example:

mAmI/buA = Either mAmI or buA. ---(1)

kaHa\_kara/kisa\_dina = kaHa\_kara or kisa\_dina ---(2)

However, if the alternatives share a common part, there is a more compact notation

for representing them using square brackets '[ ]', for example:

naHI\_lagA/naHI\_caDhA -> naHI\_lagA[caDhA] --- (3)

However square bracket notation can also be used to represent simple alternatives (not having any common part) also for example:

samaya[pava] -> 'samaya' or 'pava' --- (4)

Third and final point in the notation is the inclusion of *SUnya vikalpa* for example:

pOtA\_[HuA] = Either 'pOtA' or 'pOtA\_HuA' --- (5)

here 'pOtA' may be taken as 'pOtA\_0' for understanding purpose.

Given these notation the interface should allow the post-editor to choose one of these alternatives. There are two ways provided for doing it. The first method assumes that the user does not know about the notation. In interface the user is supposed to place the cursor anywhere on the word (with multiple alternatives) and invoke a suitable command, as a result the interface will present all the alternatives in a pop-up menu. Now the user can select the appropriate one. In this method the machine analyzes the given word with the attendant notation and produces a list of alternative words. However it is a little bit slower, because the pop-up menu in text-based environment is costly with respect to time. The second method for choosing among alternatives is very fast but assumes that the user knows the notation. In this the user puts cursor on the alternative that he wishes to choose, and invokes another command, as a result the alternative with a cursor is chosen (for more detail see Appendix B).

## 5.5 Handling jO\* Construction

The 'jO\_\*' construction as stated in chapter 2 in detail has also to be handled by the post-editor. It involves supplying *vibhakti* and some rearrangement of words in the sentence. The *vibhakti* may be 'maiM', 'sE', 'nE', 'kO', or 0 (i.e. no *vibhakti*). For example the following sentence

kAma kiyA\_Hai\_jO\*\_vaHa\_ rAmmUrTi bhalA\_AdamI[acchA].

after getting *vibhakti* 'nE' will become:

kAma jisa rAmmUrTi nE kiyA Hai vaHa bhalA\_AdamI[acchA].

Similarly the sentence

```
rAma KAyA_Hai_jO*_vaHa_ plETa dho_DAlO.
```

after getting *vibhakti* 'mEM' will become

```
rAma jisa plETa mEM KAyA hE vaha dho_DAlO.
```

And finally in the sentence

```
rAmmUrti kiyA_Hai_jO*_vaHa_ kAma acchI_taraHa  
naHIM_Hai.
```

after getting 0\_ *vibhakti* will become:

```
rAmmUrti jO kAma kiyA Hai vaHa acchI_taraHa naHIM_Hai.
```

The interface provides command for dealing with each of these. The user is supposed to place cursor on the word containing 'jO\_\*' and execute a command. The type of *vibhakti* to be supplied is specified by the first letter of the *vibhakti* with the command. For example for specifying *vibhakti* 'nE' the post-editor will execute command 'ALT-JN'. Similarly commands 'ALT-JK', 'ALT-JS', 'ALT-JO' for 'kO', 'sE' and 'O' *vibhakti* respectively.

## 5.6 Miscellaneous Commands

These are ready-made commands which are made after doing field tests and taking the advice of many post-editors. These commands do things which post-editor has to do manually and quite often. The sole aim of such command is to speed up post-editing. Some of these commands may become unnecessary as the quality of output improves.

### 5.6.1 Heuristic for plurals

Most often post-editor has to convert a word into its plural form. For example *amusaaraka* will simply produce 'laDakI{ba.}', here 'ba.' indicate that it is in *bahuvachan* (plural). Now 'laDakI' in plural can become 'laDakiyAN' or 'laDakiyOM'. Which out of these is appropriate is dependent on the context. So to tackle this problem a command is made which if invoked after putting cursor on certain word will replace that word with its most commonly used form, if invoked again it will replace that word by next most commonly used form and so on. So the



post-editor can go on invoking that command until it gets the appropriate one. The disadvantage of this command is that it may generate many non-existent forms, because it is not referring to any database but applying simple grammatical rules, and more or less like a hit and trial method, but on the other hand it is very effective speed-wise. This is illustrated by following examples:

```
laDakI - laDakiyAN - laDakiyOM
bAta   - bAtEM - bAtOM
bhAvanA - bhAvanE - bhAvanEM - bhAvanOM - bhAvanAEN
```

The above example indicates that bhAvanA in first invocation of the command becomes bhAvanAEN and then bhAvanE and so on. The simple grammatical rule that is operating here is that if the word ends with 'A', replace 'A' by 'E', and if the word ends with 'E', replace 'E' with 'EM' and so on.

If the word is 'laDakA', its correct plural 'laDakE' or 'laDakOM' can be obtained by using the command once or twice, But if the word is 'bhAvanA', then its correct plural form 'bhAvanAEN' will be obtained applying the command four times, of course the later case occurs less often.

### 5.6.2 Filling words by the user

At many places the appropriate word to be placed is ambiguous and is to be decided by the user according to the context. At such places *amusaaraka* places '\*', implying that the right word will be guessed by the user. Following is one such example.

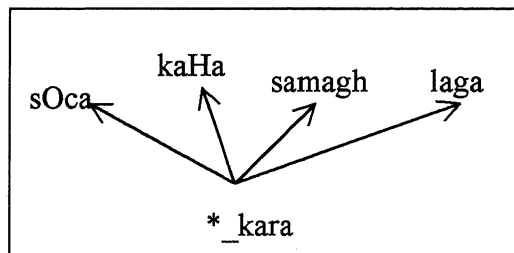


Fig. 5.1

here '\*' stands for four possible alternatives, appropriate one will be decided by the user. For this also interface has a command, the user will put cursor on the star and invoke the command. First invocation will yield 'sOca', second 'kaHa' and so on.

The order again is taken based on statistical data.

### 5.6.3 Giving Gender Number Information by the user

The gender number person information is absorbed by the commands that add *vibhakti* or perform noun-phrase agreement. If these information are in the source text, well and good (if this information is in the source text, it will be produced by the *anusaaraka* in the output as extra information within curly braces) otherwise it is convenient if the user gives these information before using executing those commands. The interface provide a way for giving this information. Its use is illustrated with following example.

*Anusaaraka* output: baDA\_ camacA

Here suppose the user has to add *vibhakti* 'sE', accordingly 'baDA' should change to 'baDI', but the information that 'camacA' has female gender is nowhere, so by default the interface will assume male gender and produce incorrect result. So before invoking the command that adds *vibhakti* the user is advised to give the gender information then it will become:

After the user gives gender info : baDA\_ camacA{strI}

Subsequent addition of *vibhakti*: baDI\_ camacA\_sE

Similarly sometimes it is convenient to give the number information beforehand.

### 5.6.4 Command for interchanging adjacent words

Because of differences in the conventions in two language it may be necessary to interchange two words in the post-edited output. For example *Telugu* text prefers to write bhAryA-bhartalu (wife-husband), it comes in the same order in the output. But in *Hindi* it should be pati-patnI (Husband-wife). Such type of cases are found very often in practice, so to tackle it in a handy way, interface has a command for interchanging two adjacent words.

### 5.6.5 Command for pulling word down

The *anusaaraka* output is available to a reader in two-line form. The first line is *Telugu* text and the second line is corresponding *anusarit* text. In the *anusarit* text

there is often a pointer to look upward in the *Telugu* text. There are two reasons for this: The *Telugu* word is unrecognized (may be because it is proper noun), or the above word may be more appropriate. Typically in such cases the user needs to copy the word from upper line. For example:

---

darvAna	atanini	Api	aDigADu.
^^^	usakO{pu.}	rOkA_[kara]	pUchA_[Hai/thA].

---

Pulling the word down in such cases is necessary, since after post-editing the *Hindi* lines will be extracted in a separate file and may be made available for untrained peoples to read. Therefore the interface provides a command for doing it. This command is intelligent in the sense that it will do all such replacements in the remaining file automatically.

### 5.6.6 Command for removing junk characters

There are characters in the *amusaaraka* notation like #, %, +, \_, ` etc. which have no meaning after the output has been post-edited. So interface provides a commands for cleaning. It removes all junk characters from the file. This command should normally be used after post-editing the file completely.

### 5.6.7 Command for inserting certain words

Certain words like 'aur', 'Hai', 'thA' etc. have their no counterpart in *Telugu* so they are not present in the *amusaaraka* output, but they are needed to be typed by the post-editor while post-editing. These are made available ready made, so that the user instead of typing them manually can get it typed quickly by click of a command. Care has been taken to keep corresponding command name intuitive, so that the user may not exert while trying to remember them.

## 5.7 On Line Thesaurus

As already mentioned that *amusaaraka* while producing output replaces a source language word, by the word that represent the nuclear sense for that word in the target

language. So by definition it is a word that triggers right meaning in the mind of reader. This is fine as far as reading is concerned but when it comes to post-editing, it has to be replaced by a more appropriate synonym. The most time consuming task for the post-editor is to think and find a suitable synonym which is used in the normal Hindi. For this purpose thesaurus is embedded with the interface. This thesaurus is extracted from the bilingual *Telugu-Hindi* dictionary which is used in the *amusaaraka*. However, if a more complete and proper thesaurus becomes available in the free domain, it can easily be incorporated with this interface. This current thesaurus is not at all adequate, however the interface provides commands for adding new words into it.

### 5.7.1 Extraction of Thesaurus

The typical bilingual dictionary entry is of the form:

```
source_lang_word1:
1.meaning11/meaning12/meaning13/2.meaning21/...,      other
information.
```

where 'meaning11' is nuclear sense corresponding to sense one of the source language word, and 'meaning12', 'meaning13' etc are its synonyms. Similarly 'meaning21' is nuclear sense corresponding to sense two and so on. Corresponding to 'source\_lang\_word1' the *amusaaraka* will produce 'meaning11/meaning21' in the output. Suppose according to the context and world knowledge, the user finds sense of 'meaning11' to be appropriate, then he will select it using interface command. Now this sense is correct and conveys meaning as far as reading is concerned, but suppose 'meaning13' was more appropriate for the post-edited text, how can a post editor get this information?

It is for this reason, the thesaurus is extracted from the dictionary, and embedded in the interface. For extracting thesaurus, first all strings of the form "meaning1/meaning2/meaning3/..." are grepped, and then for each such entry following entries are made in the thesaurus:

```
meaning1 := meaning2,meaning3,..
meaning2 := meaning1,meaning3,..
meaning3 := meaning1,meaning2,..
...
```

### **5.7.2 Using the Thesaurus**

The interface provides two ways for accessing the thesaurus. The one that is fancy, can be invoked by pressing a command after putting cursor at the word. It brings all the synonyms in a pull-down menu, and the user can make a selection. In second method the user is supposed to invoke a command by putting cursor on the right word. As a result the word will be replaced by its first synonym found in the dictionary, next invocation of the command will bring next synonym, and so on, after the last one the original word will come. Thus it is like a virtual pop-down menu with scrollbars and only one item shown at a time. The advantage with this is that, it is very fast, but it does not allow the user to see all the synonyms at once, which is desirable for making a good selection.

### **5.7.3 Enlarging the Thesaurus**

The thesaurus is very small and does not have enough entries. However the interface provides a way for adding new words into it. The user is advised, first to see the list of synonyms available in it, if the desired word is not there then (s)he should make new entry in it. These new entries are placed in the thesaurus, which is personal to the user.

There is a global thesaurus which is available to all users. Whatever entries the user makes, are kept initially into a temporary thesaurus. After every five entries (can be changed through .preferences file) the interface asks the user for confirmation of these entries. After confirmation, these entries are kept in the user's personal thesaurus. The personal thesaurus is local to the user only, but there are utilities for moving the entries of local thesaurus into global thesaurus, so that they may be available to other users also.

## 6. Conclusions

*Anusaaraka* proposes to overcome the language barrier in India by taking advantage of the relative strengths of the computer and the human; the computer takes the language load and leaves the world knowledge to the human. The human being helps the process of machine translation at two stages: pre-editing the input, and post-editing the output. Accordingly there are pre-editing and post-editing interfaces. In addition there is a reading interface to help the user in reading the output directly.

The pre-editing interface helps a person to correct mistakes and standardize the input text. In the first step it identifies the words which are unrecognized by the machine. It helps the pre-editor by presenting a set of nearest possible standard spelling words. In practice this is not a very good solution as the list presented is often long. Some research in the area of linguistic is required to find the logic behind spelling variations, and rules, if there are any, to automate it. It also helps the user in splitting the *sandhi*, and in performing the *sandhi*. Breaking *sandhi* affects the vowels at the break points. There are some heuristic rules which determine these changes. The interface tries all applicable rules, and presents the results to the user for selection. The interface does not do anything automatically and is totally controlled by the user. For example user will take the cursor at the next unrecognized word, and decide whether it is the case of dialectal spelling or of *sandhi* splitting etc. and invoke the appropriate command. Some of the things can probably be automated if linguists give sufficient rules and data.

Reading interface helps the user in comprehending the output. It controls the amount of information presented to the reader in the first instance. It presents the user with a version of output which has the best alternative for all the words with multiple alternatives. The other alternatives are available to the user if (s)he asks. It decides about the best alternative depending upon the context. It has context-information about some of the words stored in a database. This database keeps on maturing as the interface is used more and more. Further it also provides the reader with on line help on the special symbols which are used in the output to convey information. It also provides brief help on the words marked with back-quotes and detail tutorial on the words enclosed by ' | '. It also provides dictionary look-up for approximate matches,

for words which are not recognized by the machine. For foreign words which are brought as they are in the output, the interface provides *vibhakti* help. The target language (Hindi) thesaurus is also integrated with the interface. So if user have problems because he does not understand a target language word, he can consult it to see all its synonyms.

The *anusaaraka* output follows the grammar of the source language. The post-editing interface helps a person in making the output grammatically correct from the target language point of view. It provides high level commands for doing gender number person agreement, and adding *vibhaktis* (which automatically takes care of noun-phrase agreement). It provides command for converting 'jO\_\*' construction to normal Hindi construction, upon being supplied the missing *vibhakti* by the post-editor. It also makes the target language thesaurus available online. Thus the user can select the appropriate synonym for any target language word. The interface also provides a way for adding new words into the thesaurus, so that it will get matured with time. Besides these, the interface also provides certain low-level commands. These low-level commands are added after doing field tests, and upon the suggestions of many post-editors. The sole aim of these commands is to speed-up the post-editing, by providing a direct way of doing things which are required to be done most often. Typical example of such commands are: a command for interchanging two adjacent words, a command for converting a word into its plural form, a command for adding few commonly occurring words (which do not have their counterparts in the source language), a command for removing junk characters etc.

## Future Work

The additional things that reading interface can do is to keep a model of the user. While the user is reading text it analyses that text for possible difficulties. And then it checks the user's model for whatever concepts the user is already aware of, and provide intelligent help even before the user asks. Sometimes if the two or more difficulties are present in the same sentence it might warn the user for the difficulty that may be caused due to possible interaction of these. This warning may be given even when the user is already aware of each of these difficulty individually. This is longer term goal. The first step is to make an interface which provides help when the user invokes it. This first step has already been achieved in this work.

post-editor is doing and learn from it. By doing this it can build knowledge-bases needed for the second module of the machine translation. The second module of the machine translation system may do domain specific knowledge-based processing, etc. in which it may utilize world knowledge, frequency information, concordances etc. One concrete example of the sort of things it can do is the following. The Telugu word 'tO' has been mapped to two words of Hindi viz. 'sE' and 'kE\_sAtha'. In *anusaaraka* the equivalent word corresponding to 'tO' that is brought in the output is 'sE`'. The user has to convert it to 'sE' or 'kE\_sAth' according to the context using interface commands. Now the interface can store the context information in its database every time the user invoke that particular command. Later on this database can be used to automate this phenomenon. Similarly many things can be thought of. Currently at the name of learning the interface is only learning the target language vocabulary and the context information regarding some words.

Many things that were found to be mechanical have been automated. But till now everything is controlled by the user. That is user has to think and decide what has to be done and invoke the appropriate command. The interface so developed is a simple user interface, it has yet to evolve into a intelligent user interface, the prerequisite for this has however been achieved.



# Bibliography

- [1]. Natural Language Processing: A Paninian Perspective, Akshar Bharati, Vineet Chaitanya, Rajeev Sangal, Prentice-Hall of India, 1995.
- [2]. Anusaraka: A Device to Overcome the Language Barrier, V. N. Narayana, Ph.D. thesis, Dept. of CSE, I.I.T. Kanpur, 1994.
- [3]. User Interface Design Guidelines: A Tutorial Survey, T. N. Nigam, T.V. Prabhaker, Technical Report TRCS-96-243. Dept. of CSE IIT Kanpur.
- [4]. ANUSAARAKA: Machine Translation in Stages, Akshar Bharati, Vineet Chaitanya, Amba P Kulkarni, Rajeev Sangal, Vivek ISSN 0970-1618 Vol.10, No.3, July 1997.
- [5]. ANUSAARAKA: Overcoming the Language Barrier in India, Akshar Bharati, Vineet Chaitanya, Amba P. Kulkarni, and G. Uma Maheshwar Rao. In Anuvad: Approaches to translation, Rukimini Bhaya Nair (editor), Katha, New Delhi, 1998 (forthcoming).
- [6]. Linguist's Workbench- A Grammar Development Tools for Indian Languages, B.Srinivas, M.Tech thesis, Dept. of CSE, I.I.T. Kanpur, May 1991
- [7]. Sangal, Rajeev, Vineet Chaitanya and Harish Karnik, An approach to machine translation in Indian languages, Proc. of Indo-US Workshop on Systems and Signal processing. Indian Institute of Science, Bangalore, Jan. 1998.
- [8]. Sangal Rajeev, Machine Translation, 2001 (formerly Science Today), Jan. 1989.
- [9]. Sangal Rajeev, An Intermediate Language for Machine Translation: An approach based on Sanskrit using conceptual graph notation, Computer Science and Informatics (Journal of the Computer Society of India) 17, 9-21, 1987.
- [10]. Bharati, Akshar, Vineet Chaitanya and Rajeev Sangal, Anusaraka or language assessor: A short introduction, Presented at Seminar on Automatic Translation, Thiruvananthapuram, Int. School of Dravidian Linguistics, October 1993c (Also available as TRCS-93-205, Dept. of CSE, IIT Kanpur.)

- [11]. Bharati, Akshar, Vineet Chaitanya and Rajeev Sangal, Anusaraka as a device for the linguist, Presented at Dialogue on Future Linguistics in India, 22-24, Dec. 1993. Central Institute of Indian Languages, Mysore, December 1993d. (Also available as TRCS-93-210, Dept. of CSE. IIT Kanpur.)
- [12]. Bharati, Akshar, Vineet Chaitanya and Rajeev Sangal, An appropriate strategy for machine translation in India, In S.S. Agrawal and Subhash Pani (eds.) Information Technology Applications in Language Script and speech, BPB Publication New Delhi, 1994a.
- [13]. Bhanumati, B., An approach to machine translation among Indian Languages, Tech. Report TRCS-89-90, Dept. of CSE. IIT Kanpur, Dec, 1989.
- [14]. Bharati, Akshar, Vineet Chaitanya and Rajeev Sangal, Machine Translation and language barrier, In V.V.S Sarma, N Viswanadhan, B Yegnanarayana, and B. L. Deekshatulu (eds.) Artificial Intelligence and Expert System Technologies in the Indian Context, Vol. 2, Tata McGraw-Hill, New Delhi, 1991b, pp. 58-66.

# A. Roman Notation for Devanagari

Table A.1: Notation used in this thesis

अ	आ	इ	ई	उ	ऊ	ए	ऐ	ऐ
a	A	i	I	u	U	e	E	ai
	आ	ओ	औ	अं	अः	ॠ	ॡ	
	o	O	au	M	:	N	R	
		क	ख	ग	घ	ङ		
		k	kh	g	gh	ng		
		च	छ	ज	झ	ञ		
		c	ch	j	jh	ny		
		ट	ठ	ड	ढ	ण		
		T	Th	D	Dh	N		
		त	थ	द	ध	न		
		t	th	d	dh	n		
		प	फ	ब	भ	म		
		p	ph	b	bh	m		
य	र	ल	व	श	स	ष	ह	ळ
y	r	l	v	sh	s	S	H	L

**Table A.2: Internal representation in the Computer**

अ	आ	इ	ई	उ	ऊ	ऐ	ए	ऐ
a	A	i	I	u	U	eV	e	E
	ओ	ओ	औ	अं	अः	ं	ऋ	
	oV	o	O	M	H	z	q	
		क	ख	ग	घ	ङ		
		k	K	g	G	f		
		च	छ	ज	झ	ञ		
		c	C	j	J	F		
		ट	ठ	ड	ढ	ण		
		t	T	d	D	N		
		त	थ	द	ध	न		
		w	W	x	X	n		
		प	फ	ब	भ	म		
		p	P	b	B	m		
य	र	ल	व	श	स	ष	ह	ळ
y	r	l	v	S	s	R	h	IY

## B. Post-Editing Commands with Examples

**Typographical conventions:** The symbol ^ (caret) denote the place where the user is supposed to place the cursor. Following the 'caret' is the command which the user should enter. The symbol '->' indicate the result of entering the command. Commands are written in the bold and sample sentences of the computer output are kept in fixed width font. All examples are written in the roman notation given in table 2 of Appendix A. This notation has been used to represent devanagari characters in the computer.

The following examples also serves as a sample exercise. The user can open this file using post-editing interface and work through it.

1. **ALT-D** : will delete from the cursor to the end of the word.

e.g. rAma\_jAwA\_hE                      -> rAma  
   ^ALT-D

2. **ALT-P** : will delete from the beginning of the word till the cursor.

e.g. rAma\_jAwA\_hE                      -> jAwA\_hE  
   ^ALT-P

The above two commands are useful only with devanagari display since with devanagari display with the GIST terminal the vi command 'de' or 'dw' does not work.

3. **Commands for adding vibhakti:** Sometimes in the *anusaaraka* output *vibhaktis* are not present. It is the job of the post-editor to put them. Following are commands which will help in adding *vibhakti*. It should be noted that as a result of adding *vibhakti* the form of the noun may change, also if the noun has adjective attached to it, that adjective also needs to be adjusted. The post editor can do these things manually by actually typing but there are some tools to help him out.

The post editor should give the boundary of words that will be affected. If the



yaha -> isa\_kA  
^ALT-A

6. **ALT-E:** will add '\_ke' at the end of the word. E.g.

mota\_ kiwAba{swrI.} peja -> motI\_ kiwAba\_ke peja  
^ALT-E

7. **ALT-O:** will add '\_ko`' at the end of the word.

e.g. rameSa -> rameSa\_ko`  
^ALT-O

una\_ko` -> usa\_ko`  
^ALT-O

8. **ALT-I:** will add '\_kI' at the end of the word.

e.g. rameSa -> rameSa\_kI  
^ALT-I  
vaha -> usa\_kI  
^ALT-I

9. **ALT-M:** will add '\_meM' at the end of the word. E.g.

ganxA\_ wAlAba{swrI.} -> ganxe\_ wAlAba\_me  
^ALT-M  
vaha -> usa\_me  
^ALT-M

10. **ALT-S:** will add '\_se' at the end of the word. E.g.

badA\_ cammaca -> bade\_ cammaca\_se  
^ALT-S  
vaha -> usa\_se  
^ALT-S

11. **ALT-R:** will add '\_para' at the end of the word. E.g.

usakA\_ bAwOM -> usake\_ bAwOM\_para  
^ALT-R  
or usakA\_ bAwOM{swrI.} -> usaki\_ bAwOM\_para  
^ALT-R

12. **ALT-H**: will add '\_hE' at the end of the word. E.g.

jAwA.	->	jAwA_hE.
^ALT-H		
jAwA{ba.}.	->	jAwe_hEM.
^ALT-H		
jAwA{ba.swrI.}.	->	jAwI_hEM.
^ALT-H		

13. **ALT-W**: will add '\_WA' at the end of the word.

e.g. jAwA.	->	jAwA_WA.
^ALT-W		
jAwA{ba.}.	->	jAwe_We.
^ALT-H		
jAwA{ba.swrI.}.	->	jAwI_WIM.
^ALT-H		

14. **CRTL-H**: It is a general command overloaded to do many thing. Place the cursor at the beginning of the word and press CRTL-H.

ko`	->	ke_liye
_*	->	_se
_se	->	_meM
_meM	->	_jEse
_jEse	->	_sA
_sA	->	_ke_kAraNa
_ke_kAraNa	->	_ke_rUpa_meM
isa	->	ina
ina	->	yaha
vaha	->	usa
usa	->	una
se`	->	ke_sAWa
kiwanA	->	jiwanA
jiwanA	->	uwanA
uwanA	->	kiwanA
*_	->	soca_
soca_	->	kaha_
kaha_	->	samaJa_



samaJa_	->	laga_
laga_	->	soca_

The general strategy should be to place the cursor at the beginning of the word, and go on pressing CTRL-H till you get the nearest choice.

15. '=' : For this command the pre-requisite is to know the rules using which the alternatives are presented in the output. These are discussed in the section 5.4. The command is to place the cursor at the right word and press '='. Following are some examples:

acCA/KarAba	->	acCA
^=		
acCA/KarAba	->	KarAba
^=		
nahI_lagA[caDA]	->	nahI_lagA
^=		
nahI_lagA[caDA]	->	nahI_caDA
^=		
powA_[huyA]	->	powA
^=		
powA_[huyA]	->	powA_huyA
^=		

**CTRL-^:** Pulls the word (at the cursor position) from the upper line into the current line. e.g.

*input*

```

----- 3
o      uxyogaM  wAlUkU      vrAwa
eka    nOkarI   saMbaMXiwa  liKiwa
      ^CTRL-^
----- 4
o      uxyogaM  wAlUkU      vrAwa
eka    nOkarI   saMbaMXiwa  liKiwa
----- 5

```

*output*

-----3

```
o    uxyogaM    wAlUkU        vrAwa
eka  uxyogaM    saMbaMXiwa    liKiwa
```

-----4

```
o    uxyogaM    wAlUkU        vrAwa
eka  uxyogaM    saMbaMXiwa    liKiwa
```

-----5

**Note:** This command is intelligent in the sense that it will do all such replacements in the remaining file automatically.

16. '>' : This command will interchange the word at cursor with the next word.  
For example:

```
jAwA_hE mohana Gara.    /* place cursor at 'jAwA' and
    ^>                      press '>' */
mohana jAwA_hE Gara .    /* place cursor at 'jAwA' and
    ^>                      press '>' */
mohana Gara jAwA_hE .
```

17. ';' : This is used for cleaning. It removes all extra characters like #, %, +, \_ ,', etc. It should normally be used after post-editing the file completely.

18. <b>gn</b>	:	Replaces	j o *	by	jisane
<b>gs</b>	:	Replaces	j o *	by	jisase
<b>gk</b>	:	Replaces	j o *	by	jisako`
<b>gp</b>	:	Replaces	j o *	by	jisapara
<b>gm</b>	:	Replaces	j o *	by	jisame

#### 19. Building your own thesaurus:

This tool allows you to build your own thesaurus. While using this tool for post-editing you may find that certain word produced by the *anusaaraka* needs to be replace by some other suitable synonym. First you check for if the word

is in the thesaurus by putting the cursor on the word, and invoking 'tt' command. If the required word is not there in the thesaurus then you can store it in your personal thesaurus. It will be available for use in future.

**For example:** 'usane muJase cInI karja lI.'

Here you would like to replace 'karja' by 'uXAra'. Suppose 'uXAra' is not there in the thesaurus. You can store in using following commands.

**tr** : Put the cursor at the beginning of the word, and press 'tr'. It means that you want to give synonym corresponding to this word. The word will be deleted from the output. Now you type the word by which you want it to be replaced.

**tb** : After typing the new word go to the command mode and press 'tb'.

**tt** : Now whenever you want to replace other occurrences of the word, put the cursor at the word and press 'tt'.

**Example:**

```
AhvAna [rahA_]hE_jo*_vaha_
      ^tr
nimanwraNa [rahA_]hE_jo*_vaha_
      ^tb
oVka peVxxa maniRi AhvAna kAgiwaM
                        ^tt
oVka peVxxa maniRi nimanwraNa kAgiwaM
```

As a learning aid imagine following:

**tr** : Means thesaurus replace.

**tb** : Means thesaurus by (replace by).

**tt** : Means try thesaurus.

20. **ALT-R: Heuristic for plurals:** It will try to generate plural of a word. Suggestion is to keep on pressing ALT-Y until you get the required word. Hopefully it would help you.

```
e.g. (a) .  ladakI          -> ladakiyAz
            ^ALT-Y
            ix
```

(b).   ladakiyAz               -> ladakiyoM  
        ^ALT-Y

(c).   bAwa                   -> bAweM  
        ^ALT-Y

(d).   bAweM                  -> bAwoM  
        ^ALT-Y

(e).   BAvanA                 -> BAvanAez  
        ^ALT-Y

(f).   BAvanAez               -> BAvane  
        ^ALT-Y

21. Commands for giving additional information regarding words in the output, e.g. the user might needs to inform the interface that gender of a word is male or female.

- **vm** : It will append '{pu.}' at the end of the word. so that the interface can understand that given word has male gender.

e.g   camacA                   -> camacA{pu.}  
        ^vm

- **vf** : it will append '{swrI.}' at the end of the word. so that the interface can understand that given word has female gender.

e.g   cammaca                  -> cammaca{swrI.}  
        ^vm

- **vp** : It will append '{ba.}' at the end of the word, so that the interface can understand that given word is plural.

e.g.   cammaca                 -> cammaca{ba.}  
        ^vp

## 22. SENTENCE RECONSTRUCTION COMMANDS:

These commands try to reconstruct a sentence into more readable one.

**ALT-J0** :

rAmmUrwi kiyA\_hE\_jo\*\_vaha\_ kAma acCI\_waraha nahIM  
        ^ ALT-J0

rAmmUrwi jo kAma kiyA hE vaha acCI\_waraha nahIM

**ALT-JN :**

kAma kiyA\_hE\_jo\*\_vaha\_ rAmmUrwi Bala\_AxamI  
^ ALT-JN

kAma jisa rAmmUrwi ne kiyA hE vaha Bala\_AxamI

**ALT\_JK :**

saroja- gAyA\_hE\_jo\*\_vaha\_ gIwa bahuwa acCA\_hE.  
^ALT-JK

saroja- jisa gIwa ko gAyA hE vaha bahuwa acCA\_hE.

**ALT-JS :**

saroja- KAYa\_hE\_jo\*\_vaha\_ cammaca bahuwaacCA\_hE.  
^ALT-JS

saroja- jisa cammaca se KAYa hE vaha bahuwa acCA\_hE.

### 23. Support for gender number person agreement:

In Hindi the gender number and person of verb agrees with the gender number person of the noun. Since this gnp agreement is not present in other source language, it is decided to drop it in its image (i.e. the *anusaaraka* output). Further *anusaaraka* is unable to perform the gnp agreement since it does not know that verb should agree with which noun. So the post - editor is required to do it. Whatever gnp information was present in the source language is also reflected in the output. Wherever the necessary information is lacking, the user will give it. In the tools provided to support gnp agreement the user has to specify the verb and the noun with which it's agreement has to be done, and the interface will do it automatically. Following are the examples:

---

mEM kAra se jA\_rahA\_hE.  
^gb                      ^gf  
mEM kAra se jA\_rahA\_hUz.

---

If, however the output is marked for sentence terminators ('<.>') then the user may put cursor on the noun and issue 'ALT-G' command. The interface will automatically match it with the last verb of the sentence.

-----  
<s2> wuma kAra se jA\_rahA\_hE. <.>  
          ^gb                  ^gf  
<s3> wuma kAra se jA\_rahe\_ho. <.>  
-----

# C. Some Standard Components of an Machine Translation System

A basic MT System consists of an analyzer of the source language whose output is fed into the generator of the target language. Between the analyzer and the generator there is a mapper which uses bilingual dictionaries to map the source language elements to target language elements. The important components are described below.

## 1.1 WORD ANALYZER

Words in the input text are first processed by the morphological analyzer. Its task is to identify the root, lexical category, and other features of the given word. For example, for the Telugu word 'mAnava'; morphological analysis yields two possibilities: noun and verb.

1. mAnavuDu{category=noun,number=sg,case=oblique}

The above Telugu root 'mAnavuDu' means: 'mAnava' or man.

2. mAnuvu{cat=verb,TAM=infinitive,gnp=any}

The above Telugu root 'mAnuvu' means: 'ghAva\_bharanA' or heal

(gnp stands for gender-number-person, TAM for tense-aspect-modality.)

In the case of noun, number and case are shown, and in the case of the verb, TAM label and gnp are shown. Some more examples are given below.

smRti	smRti{cat=n,number=sg,case=0}/
	smRti{cat=n,number=sg,case=oblique}
vyAdhulaku	vyAdhi{cat=n,number=pl,case=ki}
agunaTlu	avvu{cat=v,TAM=jEsA,gnp=any}
jAta	jAta{cat=n,*adj_0* }/jAta_adj_n{n sg *obl* }/
	jAta_adj_m{n sg *obl* }

telipiri

telupu{cat=v,TAM=\*iti\*,gnp=non-neuter\_pl\_3}

The morphological analyzer we describe is designed to handle inflectional morphology. (Separate module would be needed for derivational morphology.) For a given word, it checks whether the word is in the dictionary. If found, it returns its lexical category (such as pronoun, post-position, noun, verb, etc.) and other grammatical features. It also tries to see whether the word can be broken up into a root and suffix. At the breakup point, some characters such as vowels may be added or deleted. It may have to try several times proposing to break the word at different points. For each proposed breakup, it looks up the proposed root in a dictionary and the proposed suffix in a suffix table. Whenever, both lookups are successful, it is a valid root and suffix.<sup>1</sup> From these information is returned regarding the root, its lexical category and the grammatical features.

If the above morphological analysis does not yield any answer, compounding or *sandhi* breaking is tried. The given word is broken up into two parts, and each part is analyzed as a proposed word. (Thus, for each of the two parts, the morphological analysis outlined above is repeated, which might again result in proposing roots and suffixes etc. for each proposed word.) This method is called propose and test method.

A large number of steps may have to be tried in the above procedure. There are ways of speeding up or eliminating some of the steps. But since each step is mechanical and small the machine can carry it out precisely and fast.

## 1.2. LOCAL WORD GROUPER

Indian languages have relatively free word-order, still there are units which occur in fixed order. In Hindi, the most important examples of these are the nouns followed by post-positions, main verb followed by auxiliaries, or compound nouns. In general, whenever there are a sequence of words that have a meaning which cannot be composed out of the meanings of individual words, they must be grouped together and the group as a whole will have a meaning. The group as a whole together with its meaning will have to be stored in a dictionary for a table. Some examples are given below. (Labels H and E specify the language of the sentence as Hindi and English, respectively, and '!'E' stands for gloss in English.)

---

<sup>1</sup>Provided they are compatible to each other, information about which is also stored in the dictionary.



H: laDakE kE liE  
 !E: boy for  
 E: for the boy  
 H: khAta calA jA raHA Hai  
 !E: eat walk go live is  
 E: going on eating (without stopping)  
 H: kAlA pAnI  
 !E: black water  
 E: rigorous imprisonment.

Local word grouping is more extensive in Hindi and other north Indian languages compared to the south Indian languages, while morphology is simpler. Thus, the two taken together (morphology and local word grouping), are likely to have the same level of difficulty cross the north and south Indian languages.

### 1.3 MAPPER USING BILINGUAL DICTIONARIES

This process involves looking up the elements of the source language and substituting them by equivalent elements belonging to the target language. For example, the root of a source language word obtained using a word analyzer is substituted by equivalent root in the target language. For example, 'Apa' would be produced in Hindi for the Telugu word 'mIru' (you). The grammatical features also need to be mapped suitably. For example, a pronoun and a noun in the source language (mIru and pustakaM) respectively are mapped to an appropriate pronoun and noun, in the target language below with the same number, person, etc.

T: mIru pustakaM caduvutunnArA?  
 @H: Apa pustaka paDha\_raHA\_[Hai/thA]\_kyA{23\_ba.}?  
 !E: You book read\_ing\_[is/was]\_Q.?  
 E: Are/were you reading a book?

(Where the labels mean the following:

T=Telugu, @H=anusaaraka Hindi, !E=English gloss, E=English.)

In the example above, the last word in the sentence is a verb and illustrates the mapping from Telugu to Hindi, morpheme by morpheme: the root is mapped to 'paDha' (read), and similarly the tense-aspect-modality (TAM) label is mapped to 'raHA\_[Hai/thA]' (is\_\*ing or was\_\*ing), which is followed by 'A' suffix which

gets mapped to 'kyA' (what) as a question marker in Hindi. Telugu leaves the tense open as: present or past, which is reflected in the output. GNP information is also shown separately in curly brackets ('{23\_ba.}' for second or third person and *bahu-vachana* (plural)).

## 1.4 WORD SYNTHESIZER

A word synthesizer is the reverse of the word analyzer. It takes a root, its lexical category and grammatical features, and generates a word. Two examples in Hindi are given below:

rAjA{cat=noun,number=pl,case=oblique}	=>	rAjAoM
king		
khA{cat=verb,number=sg,TAM=tA,gnp=fem_sg_3}	=>	khAtI
eat		

Word synthesis is a much simpler task compared to word analysis. This can usually be done directly by the given rules, without having to try various alternatives, by proposing and testing.

## 1.5 PUTTING THE COMPONENTS TOGETHER

The above components can be put together, resulting in an MT system. A sample system is described below, but there can be variations on this theme.

First, the input text in a source language is passed through its word analyzer, which analyzes each word and produces its root and grammatical features. These are fed into a local word grouper, which combines the words and produces local word groups. Second, the mapper takes the output produced so far, to replace the elements of the source language with elements of the target language. Thus, at this stage, the source language root will be changed to target language root. Third, the output of the mapper is fed into the generator of the target language, which itself might consist of local word splitter and morphological synthesizer. The output produced is the MT system output.

# D. Rules for Splitting and Performing *Sandhi*

These are some of the rules for splitting consonant *sandhi* within a word and across two words: some students are working on finding more rules regarding splitting the *sandhis*, and also rules for combining two words i.e. undoing *sandhis*. As and when these rules are ready, they may be integrated with the interface.

Some examples:

wiragaka	=> wirakka	#one word
poVga+goVttaM	=> poVggoVttaM	#Two words
pAwa+ceVppu	=> pAcceVppu	
paxi+xAkA	=> paxxAkA	
AdadaM	=> AddaM	

## The algorithm:

if an unrecognized word has

-gg-, -jj-, -dd-, -xx-, -bb-,  
-cc-, -tt-, -ww-, -pp-, -kk-,  
-rr- , -ll-, -LL- , -nn-, -NN- , -mm- etc.

then insert a vowel from [iua] between geminates

if morph recognize it as a word

output this word.

else

split the word at the right of a vowel that is inserted

if morph recognizes it as two words

output this word.

else

wherever two consonants are voiced [gg, jj, dd, xx, bb]

replace the first consonant by its voiceless counterpart [kctwp]  
morph may recognize them as two words. For example:

<b>input</b>	<b>vowel-insertion</b>	<b>replace 1<sup>st</sup> consonant-with</b>
gg	g[uia]g	k[uia]g
jj	j[uia]j	c[uia]j/w[uia]j/x[uia]j
dd	d[uia]d	t[uia]d
xx	x[uia]x	w[uia]x
bb	b[uia]b	p[uia]b

else

the word is unknown